

Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

5-2014

Physically Based Preconditioning Techniques Applied to the First Order Particle Transport and to Fluid Transport in Porous Media

Michael Rigley
Utah State University

Follow this and additional works at: <https://digitalcommons.usu.edu/etd>



Part of the [Applied Statistics Commons](#)

Recommended Citation

Rigley, Michael, "Physically Based Preconditioning Techniques Applied to the First Order Particle Transport and to Fluid Transport in Porous Media" (2014). *All Graduate Theses and Dissertations*. 2160.
<https://digitalcommons.usu.edu/etd/2160>

This Dissertation is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.



PHYSICALLY BASED PRECONDITIONING TECHNIQUES APPLIED TO THE FIRST ORDER
PARTICLE TRANSPORT AND TO FLUID TRANSPORT IN POROUS MEDIA

by

Michael Rigley

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Mathematical Sciences

Approved:

Dr. Joseph Koebbe
Major Professor

Dr. Jim Powell
Committee Member

Dr. Brynja Kohler
Committee Member

Dr. Nghiem Nguyen
Committee member

Dr. Eric Held
Committee Member

Dr. Mark R. McLellan
Vice President for Research and
Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2014

ABSTRACT

Physically Based Preconditioning Techniques Applied to the First Order Particle
Transport and to Fluid Transport in Porous Media

by

Michael Clay Rigley, Doctor of Philosophy

Utah State University, 2013

Major Professor: Dr. Joseph V. Koebbe
Department: Mathematics and Statistics

Physically based preconditioning is applied to linear systems resulting from solving the first order formulation of the particle transport equation and from solving the homogenized form of the simple flow equation for porous media flows. The first order formulation of the particle transport equation is solved two ways. The first uses a least squares finite element method resulting in a symmetric positive definite linear system which is solved by a preconditioned conjugate gradient method. The second uses a discontinuous finite element method resulting in a non-symmetric linear system which is solved by a preconditioned biconjugate gradient stabilized method. The flow equation is solved using a mixed finite element method. Specifically four levels of improvement are applied: homogenization of the porous media domain, a projection method for the mixed finite element method which simplifies the linear system, physically based preconditioning, and implementation of the linear solver in parallel on graphic processing units. The conjugate gradient linear solver for the least squares finite element method is

also applied in parallel on graphics processing units. The physically based preconditioner is shown to perform well in each case, in relation to speed-ups gained and as compared with several algebraic preconditioners.

(151 pages)

Key Words: Preconditioning, Particle Transport, Fluid Transport

PUBLIC ABSTRACT

Solving linear systems is at the heart of many scientific applications from the Pre-Algebra's student solving for x and y for basic geometry problems to the computational scientist solving billions of equations with billions of variables for weather forecasting, modeling fusion reactions, or web search algorithms. In this study we look at improving the efficiency of solving large linear systems that result from two applications. The first includes linear systems that result from solving differential equations for the movement of atomic particles in particle emitting, void, and absorbing regions. The second includes solving linear systems that result from solving differential equations for the flux of fluid in porous media. In both cases we employ methods of improving the linear solvers, called preconditioning, to improve the efficiency of the linear solvers. In both cases the preconditioning significantly improves the efficiency of the linear solver. These methods are also tested in parallel on graphic processing units using CUDA.

ACKNOWLEDGMENTS

This work was funded in part by a National Physical Science Consortium Fellowship sponsored by Sandia National Laboratories and by a graduate student stipend by Utah State University.

I would like to express my sincere appreciation to my advisor, Dr. Joseph V. Koebbe, for his guidance and insight throughout my program of study and during the presentation of this thesis. Additionally I would like to thank him for his willingness to take me on as a student two years into my program of study after I had decided to change the direction of my research after serving an internship at Sandia National Laboratories.

I would also like to thank my committee members for their suggestions and criticisms.

I would also like to thank my mentor at Sandia National Laboratories, Dr. Clifton Drumm, for the opportunity to work at Sandia and for his continuing collaboration.

Last, but definitely not least, I would like to thank my wife, Kimberly, for her continual support throughout my studies, research, and internships.

MICHAEL CLAY RIGLEY

CONTENTS

	Page
ABSTRACT	ii
PUBLIC ABSTRACT	iv
ACKNOWLEDGMENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	x
LIST OF ALGORITHMS	xii
1 INTRODUCTION	1
2 PHYSICALLY BASED PRECONDITIONING FOR THE FIRST ORDER PARTICLE TRANSPORT IN VOID AND HIGH SCATTERING REGIONS	8
2.1 First and Second Order Transport.....	10
2.2 FEM	14
2.3 FEM First Order Implementation	18
2.3.1 LSFEM.....	20
2.3.2 DFEM	23
2.3.2.1 Upwind Differencing	24
2.4 Linear System from FEM	25
2.4.1 Iterative and Direct Methods	27
2.4.1.1 Direct Methods.....	28
2.4.1.2 Note on Machine Precision and Computational Cost.....	32
2.4.1.3 Iterative Methods	33
2.4.2 Development of CG/BICGSTAB Methods	36
2.4.2.1 CG Method.....	36
2.4.2.2 BICGSTAB Method	42
2.4.3 Preconditioning and Iterative Methods.....	43
2.4.3.1 Preconditioned CG/BICGSTAB.....	45
2.4.3.2 Description of Physically Based Preconditioner.....	48
2.5 Numerical Results and Discussion.....	49
2.5.1 1D Source Void Problem - Reed Problem.....	50
2.5.2 2D Square Source Void Problem.....	58
2.5.3 Scattering Ratio.....	63
2.6 Conclusions and Future Work	64
3 PHYSICALLY BASED PRECONDITIONING FOR THE MIXED FINITE ELEMENT METHOD APPLIED TO A HOMOGENIZED FORM OF THE FLOW EQUATION IN POROUS MEDIA	67

3.1	Fluid Flow in Porous Media	71
3.2	MFEM Approximation	72
3.2.1	Homogenization.....	73
3.2.1.1	Homogenization Implementation.....	75
3.2.2	Projected Mixed Finite Element Method.....	78
3.2.2.1	Projection	80
3.2.3	Physically Based Preconditioner.....	84
3.2.4	Parallel Implementation on GPU	86
3.3	Numerical Results and Discussion.....	86
3.3.1	Homogenized Domain	87
3.3.2	Quarter 5-Spot Source-Sink Problem	88
3.3.3	Single Phase Flow Problem	92
3.3.4	1D Flux Varied Transmissivity Problem	93
3.4	Conclusions and Future Work	96
4	PRECONDITIONING FOR FINITE ELEMENT METHODS APPLIED TO FIRST ORDER PARTICLE TRANSPORT AND TO FLUID TRANSPORT IN POROUS MEDIA IMPLEMENTED IN PARALLEL ON GPUS	98
4.1	GPU Computing.....	102
4.2	Preconditioned Conjugate Gradient Method in CUDA	102
4.3	Source Void LSFEM First Order Particle Transport Results	104
4.4	Projected MFEM Source Sink Fluid Transport Results	107
4.5	Conclusions and Future Work	111
5	CONCLUSION	113
	REFERENCES.....	114
	APPENDICES.....	122
	APPENDIX A - CUDA TUTORIAL.....	123
A.1	Installing CUDA in Windows.....	123
A.2	Running the LSFEM on CUDA.....	124
	APPENDIX B - USERS MANUALS.....	128
B.1	Users Manuals for Particle Transport Code.....	128
B.1.1	1D LSFEM Toolbox	128
B.1.2	1D DFEM Toolbox	130
B.1.3	2D LSFEM Toolbox	132
B.1.4	2D DFEM Toolbox	134
B.2	Users Manuals for Fluid Transport Code.....	136
B.2.1	2D and 3D MFEM Toolboxes	136
	VITA.....	137

LIST OF TABLES

Table	Page
2.1 Cross Section Data for Reed Problem	51
2.2 Results of CG Method on the LSFEM for Reed Problem	53
2.3 Results of PCG Method on the LSFEM for Reed Problem.....	53
2.4 Results of BICGSTAB on the DFEM for Reed Problem	54
2.5 Results of PBICGSTAB on the DFEM for Reed Problem.....	54
2.6 Comparison of Preconditioners on the LSFEM for Reed Problem	57
2.7 Comparison of Preconditioners on the DFEM for Reed Problem	57
2.8 Cross Section Data for Square Source Void Problem	59
2.9 Results of Preconditioning on the LSFEM for Square Source Void Problem..	61
2.10 Results of Preconditioning on the DFEM for the Square Source Void Problem.....	61
2.11 Comparison of Preconditioners on LSFEM for Square Source Void Problem	62
2.12 Comparison of Preconditioners on DFEM for Square Source Void Problem..	62
2.13 Comparison of Preconditioners for LSFEM with varying Scattering Values ..	63
3.1 Results of Preconditioning on MFEM 2D Source Sink Problem	90
3.2 Results of Preconditioning on MFEM 3D Source Sink Problem	91
3.3 Results of Preconditioning on MFEM 2D Single Phase Flow Problem.....	93
3.4 Results of Preconditioning on MFEM 3D Single Phase Flow Problem.....	94
3.5 Results of Preconditioning on MFEM 2D Varied Transmissivity Test.....	95
4.1 Parallel Results of Preconditioning on the LSFEM for Reed Problem	106

4.2	Parallel Results of Preconditioning on the LSFEM for Square Source Void Problem.....	106
4.3	Comparison of Parallel and Serial Results for LSFEM on Reed Problem	107
4.4	Comparison of Parallel and Serial Results for LSFEM on Square Source Problem.....	107
4.5	Parallel Results of Preconditioning on the MFEM 2D Source Sink Problem ..	109
4.6	Parallel Results of Preconditioning on the MFEM 3D Source Sink Problem ..	109
4.7	Comparison of Parallel and Serial for MFEM 2D Source Sink Problem	110
4.8	Comparison of Parallel and Serial for MFEM 3D Source Sink Problem	110

LIST OF FIGURES

Figure	Page
2.1 Continuous Basis Functions.....	16
2.2 Discontinuous Basis Functions	19
2.3 Critical Role of Linear Systems in Science	26
2.4 Depiction of Iterative Solution Method [37]	34
2.5 Example of Steepest Descent Method [40].....	39
2.6 Example of Optimal Steepest Descent.....	40
2.7 Example of Skewed Steepest Descent	40
2.8 Example of Matrix Preconditioning for Particle Transport	48
2.9 Geometry of Reed Problem	50
2.10 Scalar Flux for the LS Method on the Reed Problem with $m = 16, n = 800$	52
2.11 Scalar Flux for the DFEM on the Reed Problem with $m = 16, n = 80$	52
2.12 Scalar Flux for the DFEM Without Preconditioning on the Reed Problem with $n = 240, m = 8$ and 32.....	55
2.13 Geometry of Square Source Void Problem	58
2.14 Scalar Flux for the Square Source Void Problem Using the LSFEM.....	59
2.15 Scalar Flux for the Square Source Void Problem Using DFEM	60
2.16 Resulting Flux Along Line $x = 5.625$ for Square Source Void Problem	60
2.17 Flux Along Line $x = 5.625$ for Several Scattering (σ_s) Values	64
3.1 Depiction of Porous Media	71
3.2 Example of Periodic Two Phase Flow Structure	74

3.3	Linear Example of Two Phase Flow Structure	75
3.4	Example of Matrix Preconditioning for Fluid Transport.....	85
3.5	Diagram of Repeated Pattern for Homogenized Problem	87
3.6	Resulting Pressure for MFEM Source Sink Problem with $m = n = 33$	89
3.7	Resulting Pressure for MFEM Single Phase Flow Problem with $m = n = 33$...	92
4.1	Resulting Flux for the Reed Problem Using the Continuous LSFEM in CUDA with $m = 16, n = 1600$	104
4.2	Resulting Flux for the Square Source Void Problem Using the Continuous LSFEM in CUDA for S10.....	105
4.3	Resulting Pressure for MFEM Source Sink Problem in CUDA with $m = n = 45$	108
A.1	Screen Shot of Results of CUDA Conjugate Gradient Method.....	124
A.2	Screen Shot of Results of LSFEM in CUDA	126

LIST OF ALGORITHMS

Algorithm	Page
2.1 Conjugate Gradient Method.....	41
2.2 Biconjugate Gradient Stabilized Method.....	42
2.3 Preconditioned Conjugate Gradient Method	46
2.4 Preconditioned Biconjugate Gradient Stabilized Method.....	47

CHAPTER 1

INTRODUCTION

Linear systems are an essential part of nearly all numerical techniques for solving differential equations and differential equations are an essential part of nearly all scientific applications. Efficiently solving linear systems thus becomes an essential part of nearly all scientific applications. Many methods have been developed for solving linear systems from a basic algebra student's substitution techniques for solving a set of two equations with two variables for basic geometry to a scientist's parallel algebraic multigrid techniques for solving a set of several million equations with several million variables for computation fluid dynamics using graphics processing units [1]. All these methods revolve around the first equation explored by a beginning linear algebra student:

$$Ax = b \tag{1.1}$$

where A is an $n \times n$ real square matrix, and x and b are $n \times 1$ vectors. Many linear system solution methods use a technique called preconditioning.

Preconditioning or rather preconditioning a linear system refers to various methods of making it easier to solve the above equation. This can generally be represented by pre-multiplying the above equation with a matrix M such that the system

$$MAx = Mb \tag{1.2}$$

is easier to solve than the original system. If such preconditioning can be done efficiently and well, it can greatly speed up the methods for solving the linear system which greatly speeds up the differential equation solvers for scientific applications. The most common

preconditioners are algebraic in nature. Algebraic preconditioners depend solely upon the matrix A and its basic structure. For instance, consider the following decomposition of the matrix A :

$$A = D + L + U \quad (1.3)$$

where D is the diagonal of A , L is the lower triangular part of A , and U is the upper triangular part of A . Several methods are derived from this decomposition including the Jacobi method, the Gauss-Seidel method, and Successive Over-Relaxation (SSOR) [2]. These and other methods are explained in more detail in Chapter 2. In this research, we look at improving, by physically based preconditioning, the efficiency of the linear system solution methods for two particular applications, first order particle transport and fluid flow in porous media.

This research began during a summer internship at Sandia National Laboratories to seek improvements on the electron-photon transport code SCEPTRE [3]. Particle transport, sometimes called radiation transport, neutron transport, photon transport, etc., models represent the interactions of small atomic particles (neutrons, photons, etc.) to determine the overall effect on various materials. Many codes, including SCEPTRE, had been written from the second order formulation of the Boltzmann transport equation [1] [4] [5] [6]. In this study we looked at solution methods for the first order formulation of the transport equation.

The first order formulation of the transport equation has been studied previously including [7] and [8]. In [8], a discontinuous finite element method is used to solve the transport equation. In [7], a least squares continuous finite element method is used. In this study we use two methods for solving the first order formulation including a

discontinuous finite element method similar to that in [7] and the least squares method from [8]. We extend the results of these methods by adding physically based preconditioning to the linear system solvers.

The linear solvers used for these systems include the conjugate gradient and biconjugate gradient stabilized methods. These are two common iterative methods that are search algorithms. They start with an initial guess vector x_0 and seek to improve that guess by searching in particular directions that are determined by the linear system matrix A . The conjugate gradient method is for symmetric positive definite matrices and the biconjugate gradient stabilized method is for any invertible matrix.

The physically based preconditioner is applied to each of these linear solvers with good improvements in efficiency in each case. The term physically based preconditioner used here refers to the fact that the preconditioners used in this study are based on the physical nature or physics of the problem being studied. In the case of particle transport, the linear system structure has a block structure due to the various angles that a particle may scatter when encountering a given point in a material. The preconditioner for the linear systems from the finite element methods for the first order transport equation is derived from the equation itself. More specifically, the preconditioner used is the system matrix that would be obtained if there were no scattering present. For each of the specific problems studied, the physically based preconditioner is compared with several algebraic preconditioners including some of those mentioned earlier like the Jacobi method and successive over-relaxation.

The second major section of this research entailed seeking improvements in efficiency for in modeling fluid flow in porous media. In [9], solutions were explored

using the mixed finite element method on the simple flow equation. For simple flows where the speed or transmissivity of the flow is determined solely by the coordinate directional flows, mixed finite element methods result in a simplified linear system structure. This structure is explained in more detail in Chapter 3. Examples of porous media flows include water flowing through a sponge, heat diffusing through asphalt, or extracting oil from underground reservoirs. In each case, the pores, or pockets within the material allow a different level of fluid flow than the material surrounding the pores. This dual speed or dual transmissivity flow results in a flow equation that, when solved using the mixed finite element method, has a large full linear system that does not lend itself directly to the simplified linear system structure of the mixed finite element method.

In this research we look at four levels of improvement for fluid flows in porous media. The first two have been explored previously which include homogenization and a projection method for the mixed finite element method applied to the flow equation. In [10], the method of homogenization is applied to the simple flow equation in a porous media domain. Put simply, the method of homogenization is a way of simplifying or averaging a porous media structure into a simpler homogenized structure. For example, we can consider a porous medium where the transmissivity in the main surrounding material is ten and the transmissivity in the pores is one. Depending on the amount of surrounding material compared to the amount of pore material, the approximate averaged or simplified transmissivity obtained by homogenization could range anywhere between one and ten. As seen in [10], this average is often related to harmonic, geometric, and arithmetic means. This homogenization allows the initial flow equation to be solved on a much coarser scale greatly reducing the computational time. However, the resulting

transmissivity does not depend solely on the flow in the coordinate directions so that the simplified matrix structure of [9] cannot be used directly.

In [11] the simplified linear system structure of the mixed finite element from [9] was extended using a projection method to address flow equations that result from homogenization of porous media flows where the transmissivity is not determined solely by the flow in the coordinate directions. In this research we extend the results of [11] in two ways. First, we improve the efficiency of the projection method by adding physically based preconditioning to the conjugate gradient linear solver of [11], and secondly we solve the linear system and apply the preconditioning in parallel using graphics processing units.

The physically based preconditioner for the projected homogenized flow equation is based on the transmissivity. The transmissivity for the projection method is broken up into its diagonal and off-diagonal components and the off-diagonal component is projected onto the diagonal component resulting in a simplified matrix structure similar to that in [9], but which structure is still a full matrix. The physically based preconditioner is the solution obtained using only the diagonal component of the transmissivity. This preconditioner is applied for several flow problems and for varying levels of off-diagonal transmissivity. In each case the preconditioner improves the efficiency of the linear solver for the projected method applied to the homogenized flow problem. This preconditioner is also compared with several algebraic preconditioner including some of those mentioned above.

The fourth level of improvement on solving the flow problem, and the third and final section of this research was applying the linear solver for the projected mixed finite

element method for the homogenized flow equation in parallel on graphics processing units using CUDA. Scientific computing on graphics processing units (GPUs) is a relatively new field. Graphics cards were originally created and driven by the video game industry as 2D display accelerators offering hardware assisted bitmap operations [12]. Over time, the benefit of GPUs was noticed and utilized in scientific applications including medical imaging, computational fluid dynamics, and environmental science [12]. GPUs have recently been utilized for both particle and fluid transport problems, [13] [14] [15] [16], as well as general preconditioning methods for linear solvers, [17] [18]. In this work we apply the conjugate gradient linear solvers of the particle and fluid transport problems together with their preconditioners on GPUs using CUDA. The physically based preconditioners perform well in parallel and, as above, are compared with several algebraic preconditioners.

In summary, two major fields of study are studied here, first order formulations of particle transport and fluid transport in porous media. Each is extended by using physically based preconditioning to improve the efficiency of the linear solvers. These solvers are also run on GPUs using CUDA where the physically based preconditioner also performs well. The remainder of the paper will be organized as follows. Chapter 2 will introduce the first area of research, particle transport, and show some of the implementation details of the finite element method, specifically the least squares and discontinuous finite element methods. Some background will also be given for linear system solvers including direct and iterative solvers and the formulation of the conjugate gradient and biconjugate gradient stabilized methods. The chapter will conclude with results of the physically based preconditioner on one and two dimensional problems.

Chapter 3 will introduce the second area of research, fluid flow in porous media and give greater details on the four levels of improvement for the mixed finite element method applied to the flow equation mentioned above: homogenization, the projection method, physically based preconditioning, and implementation of the linear solver in parallel on GPUs. The fourth level of improvement will be explored in more detail independently in Chapter 4 where some background will be given in GPU computing and the implementation of the linear solver in CUDA. Chapter 2 through 4 will each contain discussion, conclusions, and possible future work and a summary conclusion will be given in Chapter 5 followed by the appendices which will give a brief tutorial for implementing the linear solvers and physically based preconditioners in CUDA and users manuals for the codes discussed in this paper.

CHAPTER 2

PHYSICALLY BASED PRECONDITIONING FOR THE FIRST ORDER PARTICLE TRANSPORT IN VOID AND HIGH SCATTERING REGIONS

This research began as part of two internships in the Science of Extreme Environments Research Institute (SEERI) at Sandia National Laboratories (SNL). In the Radiation Effects Department one of their research and development projects is the code Sandia Coupled Electron-Photon Transport for Radiation Effects (SCEPTRE). SCEPTRE employs several second-order formulations of the Boltzmann transport equation including the even-odd parity flux (EOPF) equations and the self-adjoint angular flux (SAAF) equations. Discrete-ordinates and finite element methods are applied to the second order formulations of the transport equation and yield a linear, sparse, block-matrix system that is symmetric positive definite. These and further physical and mathematical explanations of the SCEPTRE code can be found in [3].

The first internship was focused on decreasing the run time of the conjugate gradient linear system solver for the finite element method of the Boltzmann transport equation within SCEPTRE by preconditioning the linear system resulting from the finite element method [19]. SCEPTRE utilizes the parallel linear solver package TRILINOS as well as a multi-level algebraic preconditioning package ML. Details on TRILINOS and ML can be found in [20] and [21]. As an intern, I tested the various parameters within the standard preconditioners of ML to precondition the linear system within the conjugate gradient linear solver of SCEPTRE. Full details of these tests can be found in [19].

The second internship, and the continued PhD research, was focused on developing new finite element methods and preconditioners for linear solvers for the first order transport equation [22]. Second order formulations of the Boltzmann transport equation, like those used within SCEPTRE are popular because they produce symmetric positive definite linear systems of equations which are amiable to powerful solution techniques such as the conjugate gradient method and can be readily solved on massively parallel systems using existing codes like TRILINOS [3]. The EOPF and SAAF can be found in multiple papers (see [4], [5], and [6] for examples). One downside to the second order methods is that they break down in regions with voids (i.e. vacuums). There have been several methods devised to overcome this problem, one of which is described in [23] where gradually decreasing values of the scattering cross-section were used to approach the solution in the void regions. In this research, we instead are looking to make improvements on first order methods which do not break down in voids.

Finite element methods applied to the first order equation generally result in non-symmetric and non-positive definite systems. This adds to the memory and computation needed to solve the linear systems. Because of this, preconditioning techniques were developed and applied to finite element solution methods of the first order transport equation. Results of this study are shown below.

We will first give a brief introduction into first and second order particle transport as well as a brief introduction to solving differential equations with finite element methods and look at the implementation of two methods for solving the first order transport equation, specifically a least squares finite element method (LSFEM) and a discontinuous finite element method (DFEM). Other methods that were tried will also be discussed. We

will then give a brief introduction into solving linear systems resulting from finite element methods including direct and iterative methods and then look specifically at the iterative methods used in conjunction with the LSFEM and DFEM which are the conjugate gradient (CG) method and bi-conjugate gradient stabilized (BICGSTAB) methods respectively. We will then discuss general methods of preconditioning linear systems and look at the preconditioned CG and BICGSTAB methods and follow that up with a description of the physically based preconditioner used for the LSFEM and DFEM methods.

We will then look at numerical results of the preconditioning on the LSFEM and DFEM, specifically looking at the overall speedup of applying the physically based preconditioner. We will also look at comparisons with algebraic (sometimes called blackbox) preconditioners and compare the preconditioned CG and BICGSTAB methods with direct methods. We will then summarize the results.

2.1 First and Second Order Transport

Radiation transport codes are used to evaluate the effects of radiation from various sources on materials and systems. The basic physics of radiation transport involve interactions between small particles like photons, electrons, neutrons, etc. There are many different types of particle interactions. These will not be discussed in depth here but descriptions of these interactions can be found in [3]. Here we will state more generally two types of interactions, absorption and scattering. When a particle interacts with a given material, that particle can be absorbed (i.e. lose its energy/momentum) into the

material or it can scatter off the material and have further interactions until it is absorbed or leaves the system.

Materials have various properties that can make them range from impervious to transparent to particle interactions. A simple example is light, which can travel through glass and other optically thin materials, but cannot penetrate a wall or other optically thick materials. While traveling through the glass, the particles have little interaction with the glass, whereas, when the particles hit the wall, they are either absorbed or scattered. Materials are often classified by their cross-sections, generally denoted by $\sigma = \sigma(r, E)$, where r is the position within a given material, which could be one, two, or three dimensional, and E is the energy at that position. The cross-section σ is the probability of a particle undergoing an interaction per unit path length of travel within the material [3]. In other words, the larger σ is, the less likely a particle will pass through a material, like light hitting a wall. In a vacuum, σ is zero. The total cross-section of a material is denoted $\sigma_t = \sigma_s + \sigma_a$, where σ_s is the probability of a given interaction being a scattering interaction, and σ_a is the probability of a given interaction being an absorption interaction.

Once a given material has been classified by its cross-section, the distribution of particles can be determined by the Boltzmann transport equation [3] [24]. The Boltzmann transport equation is a mathematical statement of particle balance over a differential volume [3]. The time-independent first order form is given as

$$\Omega \cdot \nabla \psi(r, E, \Omega) + \sigma_t \psi(r, E, \Omega) - \int_{D \partial B(0,1)} \int \sigma_s(r, E^*, \Omega^* \cdot \Omega) \psi(r, E^*, \Omega^*) dE^* d\Omega^* = Q(r, E, \Omega) \quad (2.1)$$

where $\psi(r, E, \Omega)$ is the angular flux of particles as position r , with energy E and traveling in unit direction Ω , σ_t is the total cross-section or probability of interaction per unit path

length at position r and energy E , σ_s is the differential scattering cross-section and gives the probability per unit path length that particles at position r with energy E^* in the unit direction Ω^* scatter into dE about E and into a cone of direction $d\Omega$ about Ω , D is the given energy spectrum and $\delta B(0,1)$ is the boundary of the unit ball.

The appropriate boundary conditions are usually given by specifying the incoming flux at an external boundary:

$$\psi(r_b, E, \Omega) = \psi_b(E, \Omega) \text{ for } \Omega \cdot n_b < 0 \quad (2.2)$$

where r_b is a position on the boundary and n_b is the outward normal to the boundary. Two of the most common boundary conditions are reflective and vacuum

$$\text{Reflective: } \psi(r_b, E, \Omega) = \psi(r_b, E, \Omega') \text{ for } \Omega \cdot n_b < 0 \quad (2.3)$$

$$\text{Vacuum: } \psi(r_b, E, \Omega) = 0 \text{ for } \Omega \cdot n_b < 0 \quad (2.4)$$

where Ω' is the adjoint or reflective directions to Ω . In one dimension, $\Omega' = -\Omega$. The vacuum condition simply states that no particles are entering the system. Other boundary conditions are also commonly used and a good discussion of boundary conditions can be found in [25].

Equation (2.1) is the first order time independent form of the Boltzmann transport equation. Examples of a time dependent form can be found in [26] and [27]. There are certain computational difficulties that arise from using the first order formulation of the transport equation. The streaming term in Equation (2.1), $\Omega \cdot \nabla \psi(r, E, \Omega)$, makes the equation non-symmetric and non-positive definite, so when applying a finite element method to the equation, the resulting linear system is also non-symmetric and non-positive definite. Lack of symmetry requires a greater storage load for a solver since only half of a symmetric matrix need be stored, and this coupled with non-positive definite

rules out many of the most efficient linear solvers. For example, Cholesky factorizations, one of the fastest direct linear solvers, and conjugate gradient methods, an efficient iterative solver, both require that a matrix be symmetric and positive definite. There are many solvers that exist for non-symmetric and non-positive definite matrices like the biconjugate gradient method (BICG), generalized minimum residual method (GMRES), and direct LU factorizations, but all of these methods require more memory and more computational time than their counterparts for symmetric and positive definite matrices. Because of this, the transport equation has often been solved from a second order formulation of the equation.

There are several second order formulations. Two of the more well-known are the Self-Adjoint Angular Flux Formulation (SAAF), and the Even and Odd Parity Formulation (EOPF). These formulations lead to linear systems that have symmetric positive definite matrices from the finite element method and therefore are easily solved by efficient solvers such as the Cholesky factorization and the conjugate gradient method. However, there are drawbacks to these solvers as well. Both the SAAF and the EOPF include terms with the inverse cross-section $1/\sigma_t$. Thus, these methods break down in regions with voids when the cross-section is zero. There are several techniques used to get around this problem. One is discussed in [23] where the EOPF is solved at decreasing levels of the total cross-section (eg $\sigma_t = 0.1$, $\sigma_t = 0.01$, $\sigma_t = 0.001$). Rather than looking for methods around the problem of voids for second order problems, this project continues the research being done to speed up the process of solving the first order equation directly, the formulation of which does not break down in voids.

As mentioned above, the first order formulation leads to matrices that are non-symmetric which require more computations to solve linear systems. Because of this preconditioning is applied to the first order solution methods to speed up the linear system solver. Below we will go through two such solution methods, the least squares finite element method (LSFEM) and the discontinuous finite element method (DFEM) together with the physical based preconditioner used to speed up the linear system solver of each method. Before doing so, we will first give a brief introduction to using finite element methods to solve differential equations.

2.2 FEM

There are two main computational methods for solving differential equations, finite difference methods and finite element methods. Both computational methods start by discretizing the domain of the problem. For example, consider the simple differential equation

$$-\frac{d^2u}{dx^2} = f \text{ in } \Omega = [0,1] \quad (2.5)$$

$$u = \begin{cases} u_0 & x = 0 \\ u_1 & x = 1 \end{cases} \quad (2.6)$$

where Ω is a given region, $\delta\Omega$ is the boundary of the region, u is a function of x , and f is some given function of x . To solve this equation by either finite element or finite difference methods, we would first discretize the domain

$$0 = x_0 < x_1 < \dots < x_n = 1 \quad (2.7)$$

for some finite positive integer n . For simplicity, assume that the distance between each point is Δx . At this point finite difference methods would proceed by discretizing the

differential operator within the differential equation to form a system of approximate equations

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} = f_i \quad (2.8)$$

for each index i between 1 and $n - 1$. We thus have $n - 1$ unknowns, u_1 through u_{n-1} , with $n - 1$ equations (recall that $u(x_0) = u_0$ and $u(x_n) = u_1$ are known) and can thus solve the following linear system to obtain an approximate solution.

$$\begin{bmatrix} -\frac{2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \cdots & 0 \\ \frac{1}{\Delta x^2} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \frac{1}{\Delta x^2} \\ 0 & \cdots & 0 & \frac{1}{\Delta x^2} & -\frac{2}{\Delta x^2} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} f_1 - u_0 / \Delta x^2 \\ \vdots \\ \vdots \\ \vdots \\ f_{n-1} - u_1 / \Delta x^2 \end{bmatrix} \quad (2.9)$$

Like many linear systems resulting from discretization of differential equations, the above system has a symmetric positive definite matrix.

Finite element methods also start with a discretization of the domain, Equation (2.7). However, rather than discretize the differential operator at each point to form a system of linear equations, the function itself is "discretized." More specifically, the solution of the differential equation is approximated by a set of simpler functions. These simpler functions are often called basis functions or test functions. One common test function is a piecewise linear function, sometimes called a hat function or Chapeau basis function, see Figure 2.1.

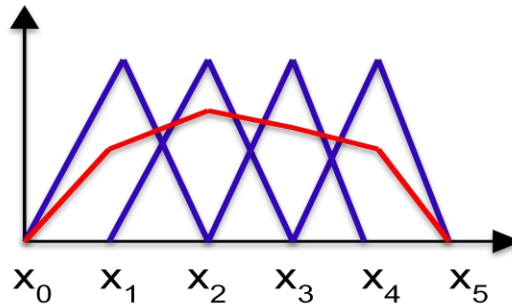


Figure 2.1. Continuous Basis Functions

These functions are of the form

$$\phi_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \quad (2.10)$$

Assigning each basis function, $\phi_i(x)$, a different coefficient we can form a piecewise linear approximation of the solution to the differential equation, Equation (2.5).

$$u(x) \approx \sum_{i=0}^n u_i \phi_i(x) \quad (2.11)$$

Thus if we can find the values of the coefficients, u_1, u_2, \dots, u_{n-1} , then we have an approximation to the solution. The finite element method proceeds by applying a Galerkin method. We will not explain the full details of the Galerkin method here. These details can be found in [28]. We will outline the basic process. To find the solution, each side of Equation (2.5) is multiplied by a test function (ie basis function) and integrated over the domain of the problem.

$$-\int_{\Omega} \frac{d^2 u}{dx^2} v dx = \int_{\Omega} f v dx \quad (2.12)$$

Then applying integration by parts to the left side of the equation, we have

$$-\left. \frac{du}{dx} v \right|_{x_0}^{x_n} + \int_{\Omega} \frac{du}{dx} \frac{dv}{dx} dx = \int_{\Omega} f v dx \quad (2.13)$$

A solution obtained in this manner is generally called a weak solution to the differential equation. If we now insert the approximation, Equation (2.11), into this equation, we have another system of linear equations

$$\int_{\Omega} \frac{d}{dx} \sum_{i=0}^n u_i \phi_i(x) \frac{d}{dx} \phi_i dx = \int_{\Omega} f \phi_i dx \quad (2.14)$$

Note that each of the test functions, $\phi_i(x)$ is zero at the boundaries, i' between 1 and $n - 1$, so we can drop the first term of Equation (2.13). We thus have $n - 1$ equations, one for each of the basis functions, ϕ_1 to ϕ_{n-1} , and can solve the system for the $n - 1$ unknown coefficients. If we once again assume that the partition of the domain has a constant width, Δx , between points, then we have the following linear system resulting from the finite element method. Note that each of the entries on the right hand side of the linear system must still be integrated. These integrals are generally completed using basic numerical integration techniques like the Trapezoid Rule or Simpson Method which can be found in any Calculus textbook.

$$\begin{bmatrix} \frac{2}{\Delta x^2} & -\frac{1}{\Delta x^2} & 0 & \cdots & 0 \\ -\frac{1}{\Delta x^2} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\frac{1}{\Delta x^2} \\ 0 & \cdots & 0 & -\frac{1}{\Delta x^2} & \frac{2}{\Delta x^2} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} \int_{\Omega} f \phi_1 dx \\ \vdots \\ \vdots \\ \vdots \\ \int_{\Omega} f \phi_{n-1} dx \end{bmatrix} \quad (2.15)$$

There are pros and cons to using either the finite difference or the finite element methods. Both are approximations and require certain constraints in order to obtain an

accurate solution. In general, the finite difference method is easier to implement. Note that no integration is needed to arrive at the associated linear system. The function only needs to be evaluated at the partition points in the domain, whereas the finite element method may require a numerical integration of the right hand side. Although the finite element method is generally more difficult to implement, it is generally considered the more robust of the two methods. This is because of the flexibility in the choice of basis functions. Above we used piecewise linear basis functions, but these can be replaced by quadratic or other basis functions to achieve a greater level of accuracy.

Particle transport problems have been solved using both methods (see [29] and [23] for examples). As mentioned above, SCEPTRE at Sandia uses finite element methods to solve second order forms of the Boltzmann transport equation. This particular research applied finite element methods to a first order form of the transport equation, focusing primarily on the linear system resulting from the finite element method. An introduction to solving linear systems resulting from differential equations will be given later, but now we will go through the implementation of the finite element methods applied to the first order transport equation.

2.3 FEM First Order Implementation

There were two main approaches of the finite element method applied to the first order transport equation, a least squares finite element method (LSFEM), and a discontinuous finite element method (DFEM). These methods differ by their choice of basis functions applied within the finite element method. The LSFEM applies test functions that are in the form of the transport operator itself. This approach when applied

through the finite element method produces a linear system that is symmetric positive definite which can be solved using fast linear solution techniques like the conjugate gradient method. The DFEM applies test functions where solutions are allowed to be discontinuous at the boundaries of the partitions of the domain.

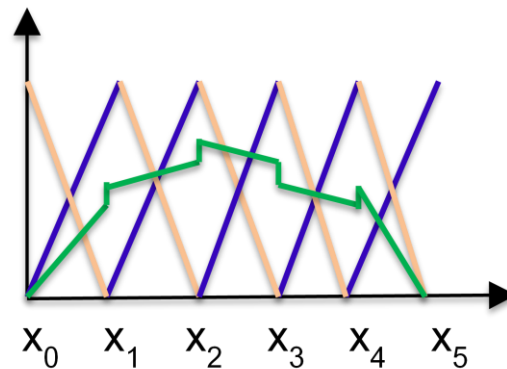


Figure 2.2. Discontinuous Basis Functions.

This is the same as taking each of the hat functions shown in Figure 2.1 and splitting each of them into two separate functions, see Figure 2.2. This method results in a matrix that is not symmetric so the system must be solved by less efficient linear system solution methods.

One other solution method was also tried using a mixed finite element method. This method has been applied in the past to the second order form of the transport equation (see [30]), and it was applied here to the first order form of the equation. The mixed finite element method is explained in more detail in Chapter 3, but the basic idea is that the solution to the transport equation, ψ , is assumed to be in a different approximation space from that of the velocity, $\nabla\psi$. This method was not pursued as far as the LSFEM and the DFEM. The linear solver run time did initially look to be faster than that of the LSFEM and DFEM, but fluctuations in the solutions seemed to indicate that the solution was not

as stable as the other methods which would have required using a much smaller step size to achieve similar accuracy. More work to obtain further explanations and better results could be done on this in the future. For this research, the focus was instead placed on increasing the performance of the linear system solvers of the LSFEM and DFEM.

We will start by going through the details of the finite element method for the LSFEM and DFEM applied to the first order form of the transport equation. We will first look at the LSFEM, then the DFEM, and then look at a description of the physical based preconditioner for each method.

2.3.1 LSFEM

We start with the first order transport equation. We will go through the details in two spatial dimensions. The results for three spatial dimensions are similar. We will use the single energy group form of the equation.

$$\Omega \cdot \nabla \psi + \sigma_t \psi(\mathbf{r}, \Omega) - \frac{\sigma_s}{4\pi} \int_{B(0,1)} \psi(\mathbf{r}, \Omega') d\Omega' = Q \quad (2.16)$$

We use this form because we are particularly interested in the performance of the preconditioning of the linear system and for multiple energy groups, the equation would be solved at each discrete level separately. This equation is first discretized in angle to obtain

$$\Omega_{m'} \cdot \nabla \psi_{m'} + \sigma_t \psi_{m'}(\mathbf{r}) - \sigma_s \sum_{m=1}^M \omega_m \psi_m = Q_m \quad \text{for } m = 1, \dots, M \quad (2.17)$$

where the scalar weights, ω_m , $m = 1, 2, \dots$, are determined by the angular quadrature.

Gauss-Legendre quadrature is used for the one dimensional case and symmetric level sets

are used for two dimensions. In two dimensions, if we define $\Omega_m = \langle \mu_m^x, \mu_m^y \rangle$, Equation (2.16) can be written in vector form as

$$(\Lambda + \sigma_t I - \sigma_s W) \vec{\psi} = \vec{Q} \quad (2.18)$$

where each of the terms are defined as follows

$$\Lambda = \begin{bmatrix} \mu_1^x & & \\ & \ddots & \\ & & \mu_M^x \end{bmatrix} \frac{\partial}{\partial x} + \begin{bmatrix} \mu_1^y & & \\ & \ddots & \\ & & \mu_M^y \end{bmatrix} \frac{\partial}{\partial y}, \quad (2.19)$$

$$W = \begin{bmatrix} \omega_1 & \cdots & \omega_M \\ \vdots & \ddots & \vdots \\ \omega_1 & \cdots & \omega_M \end{bmatrix}, \quad (2.20)$$

$$\vec{\psi} = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_M \end{bmatrix} \quad (2.21)$$

$$\vec{Q} = \begin{bmatrix} Q \\ \vdots \\ Q \end{bmatrix} \quad (2.22)$$

For this method piecewise linear basis functions were used that are defined to be 1 at a given node and zero at all other nodes. An example of the basis functions are shown in Figure 2.1, and in 2-Dimensions the Cartesian product is

$$u_{ij}(x, y) = \phi_i(x)\phi_j(y) \quad (2.23)$$

with

$$\phi_i(x) = \begin{cases} \frac{x - x_i}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \quad (2.24)$$

The angular flux is then approximated as a linear combination of the basis functions. The resulting equation then becomes

$$\sum_{j=1}^{N_y} \sum_{i=1}^{N_x} (\Lambda + \sigma_t I - \sigma_s W) \vec{\psi}_{ij} u_{ij}(x, y) = \vec{Q} \quad (2.25)$$

where the flux vector is defined as

$$\vec{\psi}_{ij} = \begin{bmatrix} \psi_1^{ij} \\ \vdots \\ \psi_M^{ij} \end{bmatrix} \quad (2.26)$$

To determine the flux coefficients we proceed with the finite element method by multiplying Equation (2.25) by a set of test functions and integrating over the domain. For the LS method, the set of test functions is given by the transport operator applied to the basis functions. In vector form we have

$$\vec{\theta}_m^{i'j'}(x, y) = (\Lambda + \sigma_t I - \sigma_s W) \vec{e}_m u_{i'j'}(x, y) \quad (2.27)$$

where \vec{e}_m is the unit vector of all zeros with a 1 in the m th position. This method was originally presented in [7] and [31]. Pre-multiplying the vector Equation (2.25) by the given test function and integrating over the domain we have a system of linear equations given by

$$\begin{aligned} & \sum_{j=1}^{N_y} \sum_{i=1}^{N_x} \int_A \left((\Lambda + \sigma_t I - \sigma_s W) \vec{e}_m u_{i'j'}(x, y) \right)^T \left((\Lambda + \sigma_t I - \sigma_s W) \vec{\psi}_{ij} u_{ij}(x, y) \right) dA \\ & = \int_A \left((\Lambda + \sigma_t I - \sigma_s W) \vec{e}_m u_{i'j'}(x, y) \right)^T \vec{Q} dA \end{aligned} \quad (2.28)$$

2.3.2 DFEM

The steps for the DFEM are similar. We will cover the basic development of the method. Further details of the method can be found in [8]. The basis functions in Equations (2.23) and (2.24) are modified according to the discontinuous scheme. In this case the functions are discontinuous at the nodes, so for each of the given elements we have a left and right function in the x -direction and a left and right function in the y -direction. See Figure 2.2 above. The result is

$$u_{ij}^{ll}(x, y) = \phi_i^l(x)\phi_j^l(y) \quad (2.29)$$

$$u_{ij}^{lr}(x, y) = \phi_i^l(x)\phi_j^r(y) \quad (2.30)$$

$$u_{ij}^{rl}(x, y) = \phi_i^r(x)\phi_j^l(y) \quad (2.31)$$

$$u_{ij}^{rr}(x, y) = \phi_i^r(x)\phi_j^r(y) \quad (2.32)$$

with

$$\phi_i^l(x) = \begin{cases} \frac{x_i - x}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ 0 & \textit{otherwise} \end{cases} \quad (2.33)$$

and

$$\phi_i^r(x) = \begin{cases} \frac{x - x_i}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ 0 & \textit{otherwise} \end{cases} \quad (2.34)$$

The set of test functions used to create the linear system are simply the basis functions themselves.

$$\bar{\theta}_m^{i'j'}(x, y) = \bar{e}_m u_{i',j'}(x, y) \quad (2.35)$$

The linear system of equations then becomes

$$\sum_{j=1}^{N_y} \sum_{i=1}^{N_x} \int_A (\bar{e}_m u_{i,j}(x, y))^T \left((\Lambda + \sigma_t I - \sigma_s W) \bar{\psi}_{ij} u_{ij}(x, y) \right) dA = \int_A (\bar{e}_m u_{i,j}(x, y))^T \bar{Q} dA \quad (2.36)$$

2.3.2.1 Upwind Differencing

When integrating the first order term of the DFEM linear system, Equation (2.36), upwind differencing is used. More details on upwind differencing can be found in [32].

We will explain the implementation details here. From Equation (2.36), integration by parts is performed on

the first order term creating a difference term and moving the derivative to the test function. That is

$$\begin{aligned} \int_A (\bar{e}_m u_{i,j}(x, y))^T (\Lambda \bar{\psi}_{ij} u_{ij}(x, y)) dA &= \int_A u_{i,j}(x, y) \left(\left(\mu_m^x \frac{\partial}{\partial x} + \mu_m^y \frac{\partial}{\partial y} \right) \psi_m^{ij} u_{ij}(x, y) \right) dA \\ &= \mu_m^x \psi_m^{ij} \phi_i^l(x) \phi_i^l(x) \Big|_x \int_y \phi_j^r(y) \phi_j^r(y) dy + \mu_m^y \psi_m^{ij} \phi_j^r(y) \phi_j^r(y) \Big|_y \int_x \phi_i^l(x) \phi_i^l(x) dx \\ &\quad - \int_A \left(\mu_m^x \frac{\partial}{\partial x} + \mu_m^y \frac{\partial}{\partial y} \right) u_{i,j}(x, y) \psi_m^{ij} u_{ij}(x, y) dA \end{aligned} \quad (2.37)$$

For upwind differencing we have the following calculations for the difference terms in x .

$$\mu_m^x \psi_m^{ij} \phi_i^l(x) \phi_i^l(x) \Big|_x = \begin{cases} \mu_m^x \psi_m^{i-1,j} \phi_i^l(x) \phi_{i-1}^r(x) \Big|_x = -\psi_m^{i-1,j} & \text{for } \mu_m^x > 0 \\ -\psi_m^{i,j} & \text{for } \mu_m^x < 0 \end{cases} \quad (2.38)$$

$$\mu_m^x \psi_m^{ij} \phi_i^r(x) \phi_i^r(x) \Big|_x = \begin{cases} \mu_m^x \psi_m^{i+1,j} \phi_i^r(x) \phi_{i+1}^l(x) \Big|_x = \psi_m^{i+1,j} & \text{for } \mu_m^x < 0 \\ \psi_m^{i,j} & \text{for } \mu_m^x > 0 \end{cases} \quad (2.39)$$

$$\mu_m^x \psi_m^{ij} \phi_i^l(x) \phi_i^r(x) \Big|_x = 0 \quad (2.40)$$

$$\mu_m^x \psi_m^{ij} \phi_i^r(x) \phi_i^l(x) \Big|_x = 0 \quad (2.41)$$

Basically, if the calculation is on the left or right, then the value is taken from the upwind direction. The difference terms in y are similar. Plugging these difference results into Equation (2.37) we can then solve Equation (2.37) as part of the linear system (2.36).

Now that we have the linear systems obtained from each finite element method, we will discuss the linear system solution techniques used to solve each linear system. We will first give a brief introduction for solving linear systems via direct and iterative methods. We will look at the specific iterative methods used in conjunction with the LSFEM and the DFEM, the conjugate gradient method and bi-conjugate gradient stabilized methods respectively. We will then give a brief introduction of preconditioning linear systems with a discussion of common preconditioners and then describe the physical based preconditioner used for each of these methods.

2.4 Linear System from FEM

Linear systems have been studied for some time now. They can be found from the beginning mathematician's pre-algebra book all the way up to the seasoned mathematician's high performance computing software. A relatively short list could include electrical networks, geometric linear programming, graph theory, games of strategy, forest management, fractals for data compression, genetics, harvesting of animal population, a least squares model for human hearing, and image processing [33]. Many techniques have been developed for solving linear systems from simple Jacobi iterations to more complex algebraic multi-grid solvers.

In this paper we are looking at two specific applications, particle transport and fluid transport. Particle transport is part of a wide variety of applications. A short list of the applications might include extreme environments for fusion research, long term effects of solar rays on satellites, and radiation in medicine like chemotherapy. Due to the small size of the particles, like protons, neutrons, and electrons, the computational domains for the models associated with particle transport often must be extremely refined for accurate results. In terms of linear systems, this means that the size of the linear system can be quite large.

Because of this, the use of traditional solution methods like Gaussian Elimination for linear systems is impractical. Instead iterative methods like the conjugate gradient method are used. We will first give a brief background of direct and iterative methods for solving linear systems. We will then look specifically at the iterative methods used in conjunction with the LSFEM and DFEM. The remainder of this section will describe preconditioning the linear systems for the iterative methods of the LSFEM and DFEM. This will include a brief introduction into preconditioning, some common preconditioners, and a description of the physical-based preconditioner used for the LSFEM and DFEM.

2.4.1 Iterative and Direct Methods

As portrayed in Figure 2.3, much of the modeling and calculations in science eventually end up in solving a linear system which generally results from a finite element or difference method for differential equations. Because of this, there have been many methods developed for solving linear systems, especially those that result from solving

differential equations. There are two general types of methods for solving linear systems, direct methods and iterative methods. We will briefly explain each type and derive some of the iterative methods used for the linear systems generated from the finite element methods above.

2.4.1.1 Direct Methods

Direct methods, as the name implies, go straight to the solution. In other words, they are algorithms that have a definite beginning and a definite end and generally after passing through the algorithm once, you obtain an approximate solution. One characterization of direct methods is that an approximate solution is obtained after a finite number of computations. Iterative methods produce a sequence of approximations of the linear system that may or may not converge. The most common direct method for solving linear systems is Gaussian Elimination. Most, if not all other direct methods are generally some variation of Gaussian Elimination.

Given a matrix A in $R^{n \times n}$, and a linear system $Ax = b$, where x and b are in R^n , Gaussian Elimination can be simply described as adding and subtracting equations within the linear system to obtain a solution to the linear system. This generally results in a linear system where the system matrix has zeros in the lower triangular half of the matrix. We will not explain here all the details of Gaussian Eliminations. These details can be found in any linear algebra book, for instance [2] [33] [34] [35]. We will show the idea of Gaussian Elimination using a simple example from pre-algebra. Consider the system below.

$$\begin{aligned} 5x - y &= 7 \\ 10x + 5y &= 4 \end{aligned} \tag{2.42}$$

which in matrix form is

$$\begin{bmatrix} 5 & -1 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} \quad (2.43)$$

Applying the methods learned in pre-algebra we can add and subtract different factors of the equations to obtain a solution.

$$\begin{array}{r} -2(5x - y = 7) \\ + \\ 10x + 5y = 4 \end{array} \rightarrow 7y = -10 \quad (2.44)$$

or

$$\begin{bmatrix} 5 & -1 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & -1 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ -10 \end{bmatrix} \quad (2.45)$$

At this point we can easily solve for y and then by substituting we can also find x .

In general, we start with a linear system $Ax = b$. Then, like in Equation (2.45), we seek to obtain a matrix in what is called an upper-triangular form, that is in the form where all the values in the lower half of the matrix below the diagonal are zero. This can be done, step by step, by pre-multiplying the linear system by a set of matrices that, like in pre-algebra, add and subtract different factors of the given equations. For the operation described in Equation (2.45) above, we have

$$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 5 & -1 \\ 10 & 5 \end{bmatrix} = \begin{bmatrix} 5 & -1 \\ 0 & 7 \end{bmatrix} \quad (2.46)$$

and

$$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 5 & -1 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & -1 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ -10 \end{bmatrix} \quad (2.47)$$

In this case, only one matrix is needed and is denoted

$$M_1 = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \quad (2.48)$$

Once the linear system is in the upper-triangular form, it can be simply solved by factoring the final equation in the linear system and back substituting. For the example above we have

$$\begin{aligned} x &= (7 + y)/5 \\ y &= -10/7 \end{aligned} \quad (2.49)$$

The example shown here is very simple, but the basic principles for larger systems are the same. Step by step we find matrices that reduce the system matrix until we reach an upper-triangular matrix. At this point we back-substitute to obtain the solution of the system.

Once the basics of Gaussian Elimination are understood, it is easy to explain many of its variants. These variants differ by the way that the matrix is transformed to upper-triangular form, or in other words, by the way that the values below the diagonal are zeroed out. Examples include Givens Rotations, Householder Transformations, Row Reduction with or without pivoting, LU Factorizations, and Cholesky Decompositions. LU Factorizations can be described as the end result of Gaussian Elimination. If we take the inverse of the product of the set of matrices used to create the upper triangular matrix, like in the example above, we can decompose the original system matrix A into the product of a lower triangular and upper-triangular matrix. For the simple system above we have

$$L = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} = M_1^{-1}, \quad (2.50)$$

$$U = \begin{bmatrix} 5 & -1 \\ 0 & 7 \end{bmatrix}, \quad (2.51)$$

$$\begin{bmatrix} 5 & -1 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 5 & -1 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} \quad (2.52)$$

or

$$LUx = b \quad (2.53)$$

The LU decomposition is especially useful if the same linear system is solved multiple times for different right-hand sides b . This is often the case for differential equations as the right-hand side is generally determined by a forcing function which can change from problem to problem whereas the differential operator, represented by A remains the same. In each case, to solve the linear system, one need only perform a forward substitution along with the back substitution mentioned before. Forward substitution uses the same idea as the back substitution, just from the top down instead of the bottom up like in Equation (2.49).

Givens Rotations, Householder Transformations, Row Reduction with or without pivoting, and Cholesky Decompositions are all different ways of forming the LU decomposition. Givens rotations are based on the trigonometric identity $\cos^2(x) + \sin^2(x) = 1$. They are what could be termed a fine-tuned elimination technique, because each rotation zeroes out only one element. This is great if there are only a few non-zero elements below the diagonal of a matrix, but gets computationally tedious for matrices with a large number of non-zero elements.

The Householder transformation zeroes out an entire column at a time and, while it is more efficient than Givens Rotations for matrices with a lot of non-zero elements, it still

is not as efficient as other methods. It is, however, rich in theoretical aspects of linear algebra and is used, among other things, in computing rank updates.

The Cholesky Decomposition is the fastest of the methods mentioned so far. It comes with restrictions, however, and can only be applied to matrices that are symmetric positive definite. A matrix is symmetric if it is symmetric about its diagonal, that is if $A(i,j) = A(j,i)$ for all i and j or more simply $A^T = A$. A matrix is positive definite if for any vector x , the product $x^T Ax$ is strictly greater than zero. If these two conditions are met, then the Cholesky Decomposition decomposes the matrix A into the form $A = G^T G$ where G is an upper triangular matrix. This is often called the square root of a matrix, which is a good analogy since only positive numbers have square roots. So with matrices, only symmetric positive definite matrices have Cholesky Decompositions.

As mentioned earlier, Cholesky Decompositions are the fastest of the direct methods. It is also the most stable. The restrictions on the Cholesky Decomposition often align well with solving differential equations since finite difference methods and finite element methods applied to elliptic differential equations result in symmetric positive definite matrices. This is not always the case, however, in which case Gaussian Elimination must be used instead.

2.4.1.2 Note on Machine Precision and Computational Cost

Earlier, it was mentioned that direct methods "generally" produce an approximate solution after a finite number of calculations. Some clarification is needed. Problems today have gotten very large requiring a lot of memory and a lot of computation. All of this is done on computers. Computers have limits to the precision of numbers that they

can store. For instance, the solution for y above, $10/7$, in decimal form is an infinite decimal, $1.42857142857\dots$. A computer cannot store an infinite number of digits, so numbers like this get rounded to some finite precision like 10^{-16} . This number is generally called the machine precision. Thus computer computations are only accurate up to machine precision. For linear systems like the one above, this is not an issue. The resulting value for x will still be very accurate. But for larger systems with millions, billions, or trillions of computations, the approximations of the computer can eventually result in large round-off errors. Because of this, direct methods can fail; that is they may arrive at a solution that is inaccurate relative to the exact solution of the linear system.

Sec 2.4.1.3 Iterative Methods

Due to the size and complexity of many problems studied today, linear systems are often solved using iterative methods. Iterative methods are used in place of direct methods like Gaussian elimination to reduce the computational cost and memory needed for large systems. Iterative methods start with an initial guess x_0 in R^n to the solution of a linear system and, iteration by iteration, produce a sequence of approximations based on application of the matrix A (see Figure 2.4).

These methods stop or converge when some measure of error or tolerance is reached. This tolerance is generally based on the norm of the residual vector $r = b - Ax$. Iterative methods used in this paper include the conjugate gradient method and the biconjugate gradient stabilized method. We will discuss these two methods in detail later, but first we will introduce some of the more common iterative methods.

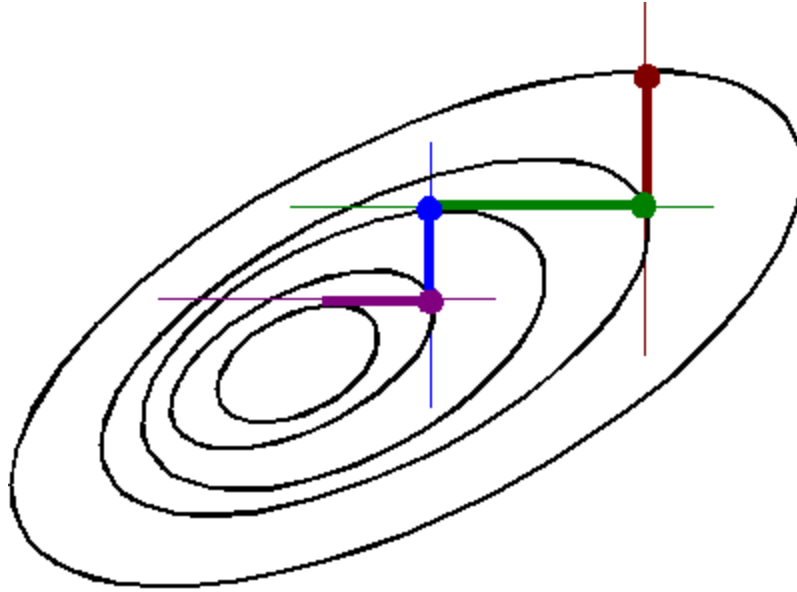


Figure 2.4. Depiction of Iterative Solution Method [36]

Several of the basic iterative methods can be classified by the general matrix splitting

$$A = D - L - U \quad (2.54)$$

where A is the system matrix, D is the main diagonal of the matrix, and L and U are the lower and upper triangular parts of the matrix respectively. With this general structure we can form several of the basic iterative methods.

$$\text{Jacobi: } x_{k+1} = D^{-1}(b - (L + U)x_k) \quad (2.55)$$

$$\text{Gauss Seidel: } x_{k+1} = (D + L)^{-1}(b - Ux_k) \quad (2.56)$$

$$\text{Successive Over Relaxation: } x_{k+1} = (1 - \omega)x_k + \omega x_k^{GS} \quad (2.57)$$

where x_k^{GS} is the k th iterate of the Gauss-Seidel method. The equations shown are the general algebraic form of these iterative method. More efficient data structures can be used when applying these equations than in storing the different matrix parts directly.

Other iterative methods include generalized minimum residual method (GMRES), where, as the name implies, the solution is found by minimizing the norm of the residual

vector. Other minimization methods are found in the conjugate gradient and bi-conjugate gradient methods. These will be discussed in detail later. One relatively newer method is the multigrid method.

Multigrid methods solve the linear system of equations at different grid sizes or levels. In the simple differential equation above, Equations (2.5) and (2.6) with the partition, Equation (2.7), you could imagine forming the linear system with $n = 64$ steps. You could also cut this in half to $n = 32$ steps or even $n = 16$ steps. As you might expect, the smaller step sizes produce less accurate approximations. However, the finer step solutions often are more susceptible to high frequency oscillations. Multigrid methods utilize benefits from both coarse and fine grid approximations by reducing both the low frequency and high frequency errors simultaneously. This is done by interpolating and extrapolating the approximations from lower to higher levels (step sizes) and higher to lower levels respectively. At each level a smoother (ie iterative solution method) is applied for several iterations.

Multigrid methods were explored in relation to this study. Indeed, the multigrid preconditioning package ML was applied to the codes within SCEPTRE to see what improvement could be made (see [19]). However, these methods were not used when developing the finite element methods for the first order transport equation. Multigrid methods can be complex involving multiple parameters, smoothers, and level strategies and the time to explore all of these possibilities was not feasible for these studies. However, based on results found in [2], multigrid methods seem to be the fastest solution method when implemented correctly and this could be a good area to look at in the future for solving the linear systems related to the LSFEM and DFEM. We will now look more

closely at the iterative methods used for the LSFEM and DFEM. We will then look at the preconditioning strategies for each method.

Sec 2.4.2 Development of CG/BICGSTAB Methods

For the LSFEM and the DFEM there were two iterative methods used, the conjugate gradient (CG) method and the biconjugate gradient stabilized (BICGSTAB) method. We will go through the derivation of the CG method and mention a few highlights concerning the BICGSTAB method.

2.4.2.1 CG Method

The CG method is an iterative method for solving linear systems. A complete derivation of the conjugate gradient method can be found in [35, pages 490-3,520-8] and [34, pages 196-203]. A less complete, but very clear method is given in [2, pages 472-474] which we will present and discuss here. Put simply it is a vector calculus problem. We consider a quadratic vector function $\phi(x)$.

$$\phi(x) = \frac{1}{2} x^T A x - x^T b \quad (2.58)$$

where x and b are vectors in R^n , and $A \in R^{n \times n}$ is a symmetric positive definite matrix. A matrix A is symmetric if $A^T = A$ or more precisely, $A(i,j) = A(j,i)$ for all i and j . A matrix A is positive definite if $x^T A x > 0$ for all x in $R^{n \times n}$. Taking the derivative of this function with respect to the vector x we have

$$\nabla \phi(x) = A x - b \quad (2.59)$$

Setting this equal to 0 we have the linear system $A x = b$. A good reference for differencing vector equations can be found in [37] and [38]. Thus the minimum of the

quadratic equation above is attained when $Ax = b$. This means that the solution to the linear system can be found by minimizing or, in other words, finding a minimum to the quadratic equation. So we can apply optimization methods to find the solution to the linear system. As mentioned above, iterative methods start with an initial guess and then seek to improve upon that guess. So for now, assume that we have some initial guess x_0 , and we have the following general equation for the next guess

$$x_{k+1} = x_k + \alpha_k s_k \quad (2.60)$$

where x_k is the current approximation, x_{k+1} is the updated approximation, α_k is the search parameter, and s_k is the search direction. The CG method is a way of finding, at each iteration k , some optimal search direction s_k and some optimal search parameter α_k . The term optimal here is loosely applied. In reality there are many ways of minimizing the function $\phi(x)$. This is because there are different ways of determining the search direction. In general, the search direction is related to the residual. We will discuss this more later. For now, we will show how to find the optimal search parameter α_k given a the previous approximation x_k and the search direction s_k . The optimal search parameter is found by taking the derivative of $\phi(x_{k+1})$ with respect to α_k .

$$\frac{d}{d\alpha_k} \phi(x_{k+1}) = (\nabla \phi(x_{k+1}))^T \frac{d}{d\alpha_k} x_{k+1} = (Ax_{k+1} - b)^T \frac{d}{d\alpha_k} s_k = -r_{k+1} s_k \quad (2.61)$$

This is combined with the following equation showing that the new residual can be written in terms of the previous residual and search direction

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha_k s_k) = r_k - \alpha_k A s_k \quad (2.62)$$

Setting Equation (2.61) equal to zero and applying the substitution, Equation (2.62), we arrive at the following value for the optimal search parameter

$$\alpha_k = \frac{r_k^T s_k}{s_k^T A s_k} \quad (2.63)$$

Thus, the optimal search parameter is a ratio involving the given residual and the given search direction. This leads back to the question of how to find the optimal search direction. As stated earlier, there are several ways of going about this.

It should be noted that part of the derivation of Equation (2.63) above showed an important intermediate result, specifically that the gradient of the quadratic function ϕ with respect to x is the negative residual vector.

$$-\nabla\phi(x) = b - Ax = r \quad (2.64)$$

Stated in other words, the direction of steepest descent of the value of the function is the residual direction. More precisely, the residual direction is only the steepest direction at that point and that if the value of the function follows just a short distance along that direction, the value may soon increase rather than decrease or at least that direction may no longer be the steepest direction. If the residual vector is always used as the search direction, then using the calculations above, we get the Method of Steepest Descent.

This idea has a very simple analogy. Picture yourself hiking down a mountain into a valley along a trail, and suppose that at some point you stop and try to figure out the fastest path to the bottom of the valley. You may try to find the steepest (probably not the safest) path in any given direction, but you'll notice that it only remains the steepest direction for a short distance before some other direction becomes the steepest.

In other words, if you always follow the steepest path, travel a short distance, and follow the steepest path again, and repeat this process, you will likely find yourself zig-zagging back and forth down the hill. This behavior was shown in Figure 2.4.



Figure 2.5. Example of Steepest Descent Method [39]

The CG method is a modification of the steepest descent method. Instead of using the residual as the search direction at every step, a modified form of the residual is used. We will first look at the idea of the modification before describing it explicitly. In terms of mountaineering, the modification can be described in the following words. Instead of always looking for the steepest path down the valley at any given point, we look for the most direct path to the bottom of the valley. If you had a perfectly circular valley or if you are standing on a sheer cliff, then the most direct path and the steepest path will be the same (see Figure 2.6).

Otherwise, the most direct path will likely be a slight modification of the steepest path (see Figure 2.7). In other words, the conjugate gradient method takes the whole mountain into consideration when determining the most direct course to the bottom of the valley rather than just the current point within the valley. In terms of matrix algebra, this means that the CG method takes the matrix A within the linear equation $Ax = b$ into consideration when determining the optimal search direction. One effect of this is that

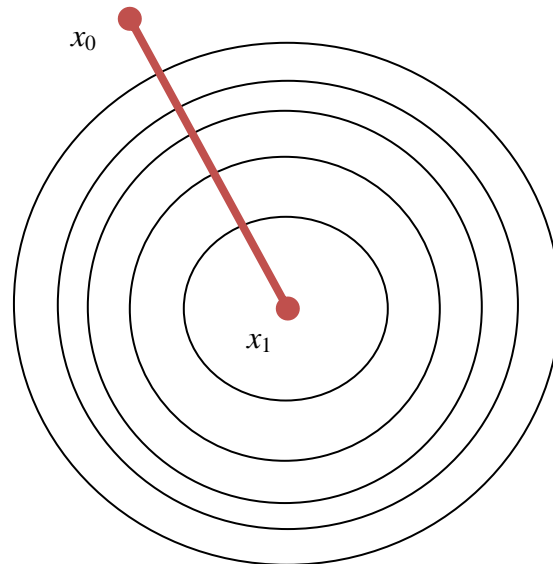


Figure 2.6. Example of Optimal Steepest Descent

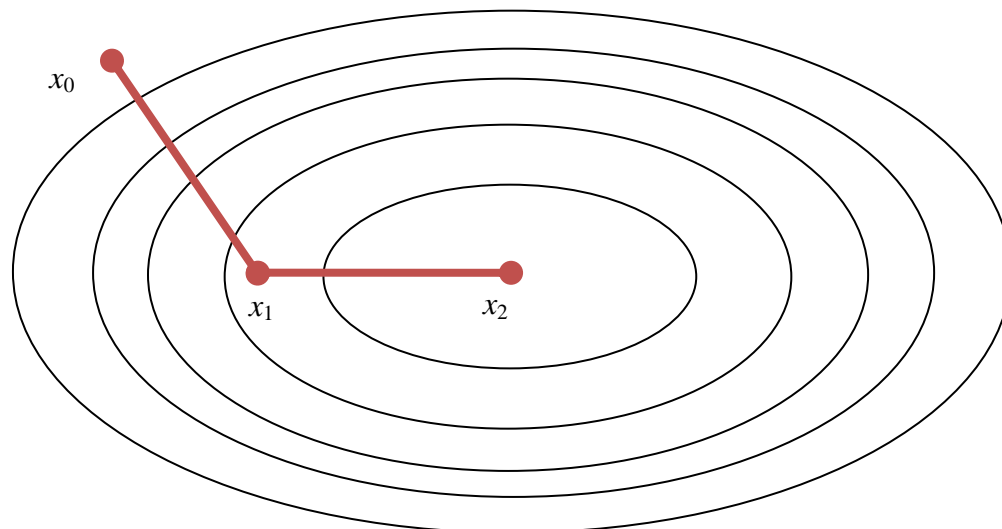


Figure 2.7. Example of Skewed Steepest Descent

future search directions are chosen in a way so as to not travel in the same direction multiple times, one of the drawbacks of the Steepest Descent Method. Specifically new

search directions are chosen so that they are A -conjugate with all previous search directions. Two vectors, u and v , are A -conjugate if

$$u^T A v = 0 \quad (2.65)$$

If we use the initial residual $r_0 = b - Ax_0$, and choose future directions so that they are A -conjugate with previous directions, a three term recurrence results [34]. In other words, only the previous two residuals are needed to make the search direction orthogonal to all the previous directions. In terms of the CG method, this adds two more equations.

$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \quad (2.66)$$

$$s_{k+1} = r_{k+1} + \beta_{k+1} s_k \quad (2.67)$$

All of this can be combined together to form the CG Algorithm for solving the linear system $Ax = b$.

Conjugate Gradient Method - See Heath [2], Page 473

A is a matrix in $R^{n \times n}$. r , b , x , and s are vectors in R^n . α , tol and β are scalars.

Algorithm 2.1 Conjugate Gradient Method

```

 $x_0$  initial guess
 $r_0 = b - Ax_0$  initial residual
 $s_0 = r_0$  initial search direction
for  $k = 0, 1, 2, \dots$ 
     $\alpha_k = r_k^T r_k / s_k^T A s_k$  search parameter
     $x_{k+1} = x_k + \alpha_k s_k$ 
     $r_{k+1} = r_k - \alpha_k A s_k$ 
    if  $\|r_{k+1}\| < \text{tol}$ , stop
     $\beta_{k+1} = r_{k+1}^T r_{k+1} / r_k^T r_k$  conjugate parameter
     $s_{k+1} = r_{k+1} + \beta_{k+1} s_k$ 
end

```

Basically, the algorithm starts with a linear system $Ax = b$, and an initial guess x_0 , and runs until some tolerance is reached, generally some measure of the residual as shown above. As mentioned earlier, this algorithm only works for symmetric positive definite

matrices. Luckily many of the linear systems resulting from solving differential equations are symmetric positive definite. For well conditioned matrices, the CG method will converge in at most n steps, where n is the size of the matrix. Generally it will converge much faster than this.

The majority of the computational time for the algorithm comes with the matrix multiplication As_k . It should be noted that this multiplication only needs to be performed once per iteration. The rest of the algorithm is made up of scalar and vector operations.

2.4.2.2 BICGSTAB Method

The bi-conjugate gradient stabilized (BICGSTAB) method is derived from the bi-conjugate gradient method (BICG). More specifically it is derived from the conjugate gradient squared (CGS) method. The details of the derivations of these methods will not be given here, but can be found in [34] [40]. Suffice it to say that these methods are similar to the CG method except that they do not require the linear system to be symmetric or positive definite. They do require the matrix to be invertible. They are based on similar conjugacy conditions as those mentioned for the CG method.

Given a linear system $Ax = b$, where we now only require A to be invertible, we have the BICGSTAB algorithm.

Biconjugate Gradient Stabilized Method - See Saad [34]

A is a matrix in $R^{n \times n}$. r , b , x , and s are vectors in R^n , α , Ω tol and β are scalars.

Algorithm A.2 Bi-Conjugate Gradient Stabilized Method

x_0 initial guess
 $r_0 = b - Ax_0$ initial residual
 $p_0 = r_0$ initial search direction
for $k = 0, 1, 2, \dots$

$$\begin{aligned} \alpha_k &= r_k^T r_0 / p_k^T A^T r_0 \quad \text{search parameter} \\ s_k &= r_{k-1} - \alpha_k A p_k \quad \text{additional search direction} \\ \Omega_k &= s_k^T A^T s_k / (s_k^T A^T A s_k) \quad \text{additional search parameter} \\ x_{k+1} &= x_k + \alpha_k p_k + \Omega_k s_k \\ r_{k+1} &= s_k - \Omega_k A s_k \\ \text{if } \|r_{k+1}\| &< \text{tol, stop} \\ \beta_{k+1} &= (r_{k+1}^T r_0 / r_k^T r_0) * (\alpha_k / \Omega_k) \quad \text{conjugate parameter} \\ p_{k+1} &= r_{k+1} + \beta_{k+1} (p_k - \Omega_k A p_k) \\ \text{end} \end{aligned}$$

There are a few things to be noted here. First note that only two Matrix multiplications need be done at each iteration, $A p_k$ and $A s_k$. The multiplication by the transpose of A need not be performed since $s_k^T A^T = (A s_k)^T$ and $p_k^T A^T = (A p_k)^T$. Similar to the CG algorithm, the algorithm stops once some tolerance is reached, generally some measure of the residual vector. The algorithms are very similar and the small differences in BICGSTAB are based on the fact that the matrix A is not symmetric.

Both the CG method and the BICGSTAB method are used in conjunction with the LSFEM and DFEM. The linear systems resulting from the LSFEM and the DFEM are solved by the CG and BICGSTAB methods respectively.

Sec 2.4.3 Preconditioning and Iterative Methods

Preconditioning linear systems has been studied for some time now and many preconditioning techniques have been created from simple Jacobi iterations to more complex algebraic multi-grid solvers. Preconditioning is related to the condition number of a matrix. Given a matrix A in $R^{n \times n}$ and some norm $\|\cdot\|$ (generally the 2-norm), the condition number is defined as

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\| \quad (2.68)$$

If the matrix A is not invertible then the condition number is defined to be ∞ . The lower the condition number (ie the closer it is to one), the more the matrix resembles the identity matrix and the easier the linear system is to solve. Consider the linear system

$$Ax = b, \text{ where } A \text{ is in } R^{n \times n}, x \text{ and } b \text{ are in } R^n \quad (2.69)$$

The preconditioner M in $R^{n \times n}$ of a linear system is generally a simpler form of A .

Preconditioning can be thought of as applying M^{-1} to each side of the original system (2.67).

$$M^{-1}Ax = M^{-1}b \quad (2.70)$$

An effective preconditioner will create a system that is easier to solve than the original system.

Preconditioners are often applied to iterative methods in order to speed up, and in some cases even obtain, convergence. The choice of the preconditioner comes down to the trade-off between number of iterations and cost per iteration. A preconditioner can significantly reduce the number of iterations, but will also increase the amount of computation at each iteration. Some common preconditioners are diagonal (also called Jacobi), block diagonal, Successive Over Relaxation, Incomplete LU Factorizations, polynomial, and multigrid [2]. These preconditioners could be called algebraic preconditioners because they depend only upon the matrix itself, not upon the problem from which the matrix resulted. These are often called black box preconditioners because they don't require any inputs other than the matrix itself. In other cases, preconditioners are created from the original problem from which the linear system resulted. These are called physically based preconditioners. Often preconditioners include some combination of both algebraic methods and physically based methods.

Examples of using both black box and physical based preconditioners are plentiful. In [19], black box preconditioners were used to optimize solution methods of second order formulations of the transport equation and in [41] a physical based preconditioner was used for the same second order formulations, which physical based preconditioner will be discussed in more detail later. There are pros and cons of each method. Black box preconditioners are nice because they are generally very easy, the one exception being multigrid preconditioners. It generally comes down to a few simple choices of parameters based on the preconditioner used. Physical based preconditioners are nice because most problems result in a particular matrix structure and this structure, when known, can be utilized to make the preconditioner more efficient. In general, a combination of both preconditioners can be used. In fact, there is no reason why the preconditioned system cannot be preconditioned itself and so on, and this is commonly done.

In this paper, a physical based preconditioner is created for the LSFEM and DFEM and is applied both by a direct method and by an incomplete LU factorization. Tests on their relative performance are done below in the results section. We will first present the preconditioned form of the algorithms.

Sec 2.4.3.1 Preconditioned CG/BICGSTAB

To solve the linear systems, Equations (2.28) and (2.36), two iterative methods were used. The LSFEM results in a linear system whose system matrix is symmetric positive definite (spd) to which the preconditioned conjugate gradient method is applied. The preconditioned CG method is very similar to the original CG method.

Preconditioned Conjugate Gradient Method - See Heath [2], Pate 474

A and M are a matrices in $R^{n \times n}$. r , b , x , and s are vectors in R^n , α , tol and β are scalars.

Algorithm 2.3 Preconditioned Conjugate Gradient Method

```

 $x_0$  initial guess
 $r_0 = b - Ax_0$  initial residual
 $s_0 = M^{-1}r_0$  initial search direction
for  $k = 0, 1, 2, \dots$ 
     $\alpha_k = r_k^T M^{-1} r_k / s_k^T A s_k$  search parameter
     $x_{k+1} = x_k + \alpha_k s_k$ 
     $r_{k+1} = r_k - \alpha_k A s_k$ 
    if  $\|r_{k+1}\| < \text{tol}$ , stop
     $\beta_{k+1} = r_{k+1}^T M^{-1} r_{k+1} / r_k^T M^{-1} r_k$  conjugate parameter
     $s_{k+1} = M^{-1} r_{k+1} + \beta_{k+1} s_k$ 
end

```

Note that the only difference is that the inner products of the residual vectors are replaced by preconditioned inner products. As mentioned above, the effect of the preconditioning is to take the system matrix into consideration when looking for the minimum solution. Since A and M are linear transformations, preconditioning can also be explained as taking the original solution space like the one in Figure 2.7 and transform the solution space itself to be more circular like that in Figure 2.6, thus making it easier for the CG method to arrive at the solution. It should be noted that the equation that we are actually solving here is the preconditioned equation of the form

$$L^{-1}AL^{-T}Lx = L^{-1}b \quad (2.71)$$

which is equivalent to $Ax = b$ where $M = LL^T$. The reason for this is to retain a positive definite system matrix so that the CG method can be applied. The algorithm formed by applying the CG method to Equation (2.71) can be rearranged to form Algorithm 2.1 [2].

The actual derivation of the preconditioned method can be found in [34] or [35].

This preconditioning is only effective (ie accurate and faster) if the preconditioner M is much easier to work with than A . Specifically it is only effective if the general system $Mx = r$ is easier to solve (ie faster) than $Ax = b$ and if M is still a good approximation of A .

The DFEM results in a linear system that is not symmetric to which a preconditioned biconjugate gradient stabilized method is applied.

Preconditioned Biconjugate Gradient Stabilized Method - See Saad [34]

$A \approx K = K_1 K_2$ are in $R^{n \times n}$. r, b, x, p and s are in R^n , $\alpha, \Omega, \text{tol}$ and β are scalars.

Algorithm 2.4 Preconditioned Bi-Conjugate Gradient Stabilized Method

```

 $x_0$  initial guess
 $r_0 = b - Ax_0$  initial residual
 $p_0 = r_0$  initial search direction
for  $k = 0, 1, 2, \dots$ 
     $\alpha_k = r_k^T r_0 / p_k^T K^T A^T r_0$  search parameter
     $s_k = r_{k-1} - \alpha_k A K^{-1} p_k$  additional search direction
     $\Omega_k = s_k^T K^T A^T K_1^{-T} K_1^{-1} s_k / (s_k^T K^T A^T K_1^{-T} K_1^{-1} A K^{-1} s_k)$  addtl srch par
     $x_{k+1} = x_k + \alpha_k K^{-1} p_k + \Omega_k K^{-1} s_k$ 
     $r_{k+1} = s_k - \Omega_k A K^{-1} s_k$ 
    if  $\|r_{k+1}\| < \text{tol}$ , stop
     $\beta_{k+1} = (r_{k+1}^T r_0 / r_k^T r_0) * (\alpha_k / \Omega_k)$  conjugate parameter
     $p_{k+1} = r_{k+1} + \beta_{k+1} (p_k - \Omega_k A K^{-1} p_k)$ 
end

```

This formulation of the preconditioned BICGSTAB method is a little more flexible than the preconditioned CG method. This flexibility results from the matrix being non-symmetric. This method can accommodate a preconditioner of the form $M = K_1 K_2$. This is very useful since the preconditioner is often decomposed into two factors like in an incomplete LU decomposition. In many cases, we can choose $K_2 = I$ in which case we can replace all the K 's in the above algorithm with M .

We can thus see the dramatic effect that a preconditioner can have on the condition of a linear system. Since the preconditioner has significantly fewer non-zeros than the original matrix it is also much easier to solve. When applied within iterative methods like the conjugate gradient and biconjugate gradient method, it adds relatively few computations to each iteration, but significantly reduces the number of total iterations needed for convergence.

2.5 Numerical Results and Discussion

We will now look at the results from using the physically based preconditioner described above. Two of the causes of computational difficulty or ill-conditioned systems in particle transport problems are voids, when the total cross-section σ_t is zero, and high scattering regions, when σ_s is very close to σ_t and problems with these characteristics were chosen for analyzing the preconditioning methods. Methods with both of these characteristics can be termed source-void problems. Each of the two finite element methods, the LSFEM and DFEM, were tested on a one dimensional source-void problem and a two dimensional source-void problem. The geometries for the problems and the results are given below.

The tests were done in MATLAB using MATLAB sparse matrix structure in conjugate gradient (CG) and stabilized biconjugate gradient methods (BICGSTAB). The two algorithms stop or converge when the tolerance or squared residual norm is 10^{-9} . Run times for all of the results were calculated using Matlab's tic and toc functions. CG and BICGSTAB methods for compressed sparse row (CSR) storage were written and tested as well, but writing the algorithms with Matlab storage allows an easier comparison

between the physical based and black box preconditioners so the results when using CSR storage are not shown here. The methods were compared against Matlab's built-in direct solver and several blackbox preconditioners including Matlab's built in incomplete Cholesky and incomplete LU factorizations. The final test looked at the performance of each of the preconditioners for the LSFEM problem as the scattering cross-section σ_s is increased from 0 to σ_t .

We will first look at the one dimensional problem and then the two dimensional problem. After looking at the general performance of the physical based preconditioner, we will then compare it with several Matlab blackbox preconditioners. These tests will then be followed by discussion and conclusions.

2.5.1 1D Source Void Problem - Reed Problem [29]

The first problem is a one dimensional source void problem given by Reed in [29]. The geometry is given in Figure 2.4 below and cross-section data given in Table 2.1. The problem contains five regions: a source, an absorber, a void, a source with scattering, and an absorber with scattering. The boundary conditions are assumed to be reflective on the left and vacuum on the right.

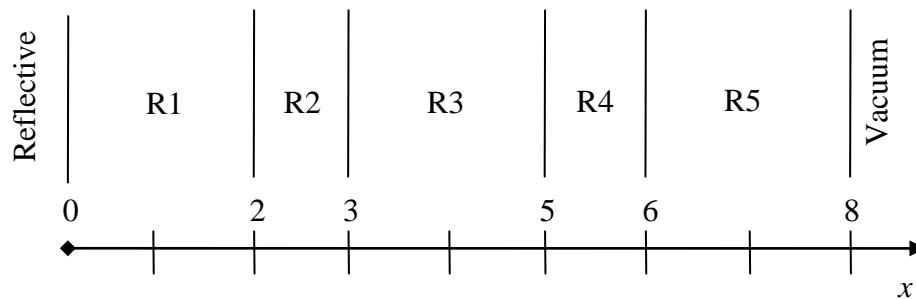


Figure 2.9. Geometry of Reed Problem

Table 2.1. Cross Section Data for Reed Problem

Region	Q	σ_t	σ_s
1	50	50	0
2	0	5	0
3	0	0	0
4	1	1	0.9
5	0	1	0.9

Both the LSFEM and DFEM perform well on this problem, but the precision needed to obtain accuracy in the void region for the LSFEM is much greater than the precision needed for the DFEM. As seen in Figure 2.10, even with the number of steps set at $N = 800$ the LS method is still linearly increasing over the void region rather than remaining constant, whereas the discontinuous method (see Figure 2.11) achieves similar if not greater accuracy with the number of steps set at $N = 80$. The problem itself is discontinuous by nature of the discontinuous interaction cross-section σ_t , so the better accuracy of the DFEM is to expected.

It should be noted that due to the choice of basis functions in Equations (2.29-2.32), the size of the matrix for the discontinuous method is $4*M*N$ where the size of the matrix for the LS method is only $M*N$. Furthermore, the DFEM matrix is not symmetric which also increases the storage. However, for the one dimensional source-void problem, the accuracy of the DFEM in the void region reduces the step size enough to make the DFEM comparable if not superior to the LS method.

LS Scalar Flux for Problem 2.5.1

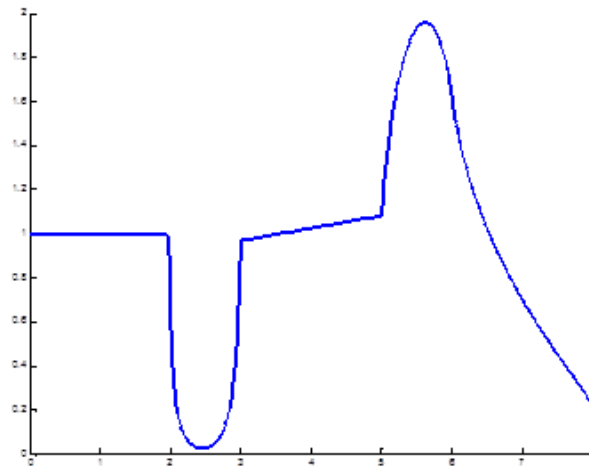


Figure 2.10. Scalar Flux for the LS Method on the Reed Problem with $m = 16$, $n = 800$

DFEM Scalar Flux for Problem 2.5.1

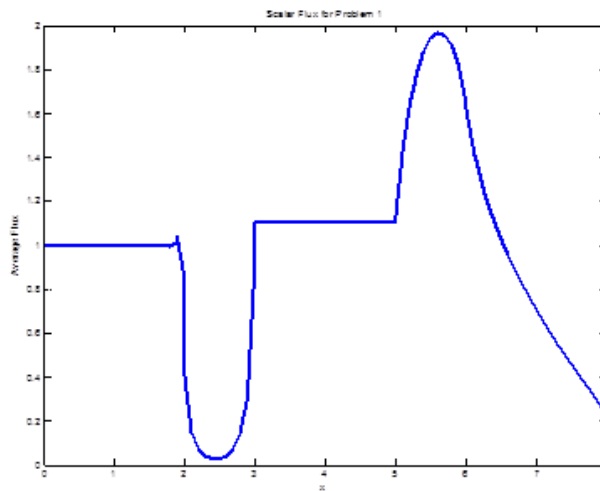


Figure 2.11. Scalar Flux for the DFEM on the Reed Problem with $m = 16$, $n = 80$

The benefit of the uncollided-flux preconditioner on each method can be readily seen. The results are shown in Tables 2.2 & 2.3 and Tables 2.4 and 2.5. The results of the conjugate gradient (CG) method for the LSFEM in Table 2.2 are compared with the results of the preconditioned CG method for the LSFEM in Table 2.3. In this case we are

using the physical based (PB) preconditioner applied in an incomplete Cholesky factorization (IC) fashion. In this case the run time is improved greatly—over a 100 times.

Table 2.2. Results of CG Method on the LSFEM for Reed Problem

CG Method	Mesh Size (n -spatial, m -direction)		
	$n = 800$	$n = 1600$	$n = 2400$
$m = 8$	1.25 s, 170 it.	3.98 s, 10396 it.	8.99 s, 15583 it.
$m = 16$	4.811 s, 8106 it.	20.64 s, 17171 it.	45.67 s, 25763 it.
$m = 32$	29.0 s, 13871 it.	115.0 s, 27979 it.	319 s, 43112 it.

Table 2.3. Results of PCG Method on the LSFEM for Reed Problem

PCG Method - PB IC	Mesh Size (n -spatial, m -direction)		
	$n = 800$	$n = 1600$	$n = 2400$
$m = 8$	0.00880 s, 20 it.	0.0175 s, 21 it.	0.0270 s, 21 it.
$m = 16$	0.0227 s, 23 it.	0.0469 s, 22 it.	0.0790 s, 22 it.
$m = 32$	0.0714 s, 24 it.	0.146 s, 25 it.	0.259 s, 26 it.

The DFEM BICGSTAB method results in Table 2.4 are compared with the preconditioned BICGSTAB method results in Table 2.5. Here the results are not only better in terms of time, in some cases still over 100 times as fast, but also in terms of convergence. Recall that the matrix in this case is not symmetric which is why the BICGSTAB method is used instead of the CG method. In two of the cases in Table 2.4 the BICGSTAB method does not converge. This is because parameters within the BICGSTAB algorithm get too small according to machine precision and result in division

by zero. As can be seen in Figure 2.12 a and b, the reason for this is due mainly to the void region in the problem. Here the physical based preconditioner is applied in an incomplete LU factorization (ILU) fashion.

Table 2.4. Results of BICGSTAB on the DFEM for Reed Problem

BICGSTAB Method	Mesh Size (n-spatial, m-direction)		
	$n = 80$	$n = 160$	$n = 240$
$m = 8$	0.0444 s, 433 it.	0.251 s, 1112 it.	DNC
$m = 16$	0.149 s, 558 it.	0.639 s, 1460 it.	1.46 s, 2487 it.
$m = 32$	0.561 s, 939 it.	3.38 s, 3153 it.	DNC

Table 2.5. Results of PBICGSTAB on the DFEM for Reed Problem

PBICGSTAB Method - ILU	Mesh Size (n-spatial, m-direction)		
	$n = 80$	$n = 160$	$n = 240$
$m = 8$	0.00148 s, 5 it.	0.00388 s, 6 it.	0.00426 s, 6 it.
$m = 16$	0.00439 s, 6 it.	0.00545 s, 5 it.	0.00785 s, 6 it.
$m = 32$	0.00681 s, 6 it.	0.0104 s, 5 it.	0.0184, 6 it.

DFEM Scalar Flux for Problem 2.5.1

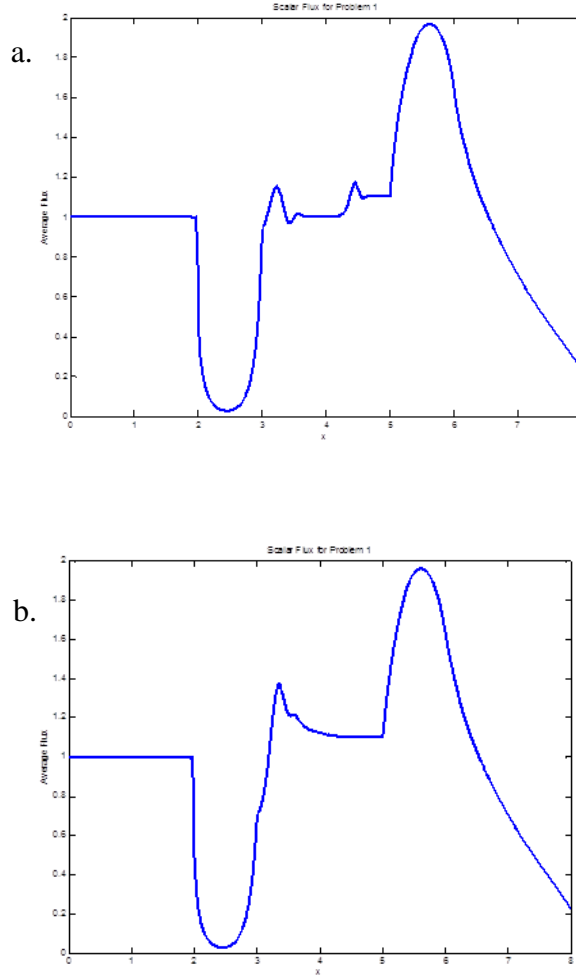


Figure 2.12 a and b. Scalar Flux for the DFEM Without Preconditioning on the Reed Problem with $n = 240$, $m = 8$ and 32

The mesh sizes were chosen for this research so that they can still be run on a single machine. The matrix sizes, therefore, were still small enough to compare with direct methods, which can often be faster than iterative methods for smaller mesh sizes, especially optimized commercial methods like those in Matlab. In Table 2.6, the physical based (PB) preconditioner is applied in two fashions, a direct Cholesky factorization fashion and an incomplete Cholesky (IC) factorization fashion. The direct and black box

preconditioners include Jacobi iterations, successive over-relaxation (SSOR), Matlab's direct Cholesky (DC) factorization, and Matlab's incomplete Cholesky (IC) factorization.

As seen in Table 2.6, the direct Cholesky (DC) outperforms the unpreconditioned CG method and Jacobi and SSOR preconditioning. This is not surprising and would be expected to change for larger mesh sizes when memory becomes more of a factor. The physical based (PB) preconditioners still outperform the direct Cholesky and are eclipsed only by Matlab's Incomplete Cholesky preconditioner. The incomplete Cholesky preconditioner is the incomplete Cholesky factorization of the original system matrix A whereas the physical based incomplete Cholesky is the incomplete Cholesky factorization on the preconditioner M . One other interesting note is that the number of iterations for the physical based incomplete Cholesky factorization and the physical based direct Cholesky, where the inner preconditioned solution is obtained through a direct Cholesky factorization, both decrease in terms of numbers of iterations rather than increase as the step size increases. This may be because the physical based preconditioner is affected more by the increase in angles (m) than in the increase in steps (n), and increasing the steps relative to the angles improves the effectiveness of the physical based preconditioner.

The results in Table 2.7 are similar to Table 2.6 with a few notable exceptions. Here SSOR is not applied within the BICGSTAB method because SSOR is only for symmetric matrices. Here the Physical Based incomplete LU factorization actually outperforms Matlab's incomplete LU factorization for some cases in terms of run time. This improvement is more readily seen in the two dimensional problem below. Also, it is interesting to note that the Jacobi preconditioner actually causes the original BICGSTAB

Table 2.6. Comparison of Preconditioners on the LSFEM for Reed Problem

PCG Method	Mesh Size (n -spatial, m -direction)		
	$m = 16, n = 800$	$m = 16, n = 1600$	$m = 16, n = 2400$
None - CG	4.811 s, 8106 it.	20.64 s, 17171 it.	45.67 s, 25763 it.
Jacobi (Diagonal)	1.77 s, 2287 it.	6.95 s, 4443 it.	15.9 s, 6688 it.
SSOR	1.20 s, 780 it.	4.66 s, 1529 it.	10.6 s, 2306 it.
None - DC	0.0474 s	0.0987 s	0.146 s
PB - DC	0.0301 s, 21 it.	0.0507 s, 20 it.	0.0729 s, 19 it.
PB - IC	0.0227 s, 23 it.	0.0469 s, 22 it.	0.0790 s, 22 it.
IC	0.0123 s, 8 it.	0.0283 s, 9 it.	0.0485 s, 9 it.

Table 2.7. Comparison of Preconditioners on the DFEM for Reed Problem

PCG Method	Mesh Size (n -spatial, m -direction)		
	$m = 16, n = 80$	$m = 16, n = 160$	$m = 16, n = 240$
None - BICGSTAB	0.149 s, 558 it.	0.639 s, 1460 it.	1.46 s, 2487 it.
Jacobi (Diagonal)	0.0983 s, 263 it.	DNC	DNC
None - DLU	0.0102 s	0.0173 s	0.0284 s
Physical Based - DLU	0.00501 s, 5 it	0.00626 s, 5 it.	0.00689 s, 5 it.
Physical Based - ILU	0.00439 s, 6 it.	0.00545 s, 5 it.	0.00785 s, 6 it.
ILU	0.00424 s, 4 it.	0.00566 s, 4 it.	0.00712 s, 4 it.

method not to converge. Slight changes to the original system can be the difference between converging and not converging. One reason for non convergence is when the matrix has large complex eigenpairs and that may be the case here [42].

In both the LSFEM and DFEM cases, the physical based preconditioner performs well in reducing the run time for solving the linear systems association with the first order formulation of the transport equation and are comparable to current black box preconditioners. In the two dimensional problems we will see even greater improvements.

2.5.2 Square Source Void Problem

The two methods were also tested on the square source void problem of Watanabe and Maynard in [23] [41] [43]. The geometry is given in Figure 2.13 below and cross-section data given in Table 2.8. The problem contains three regions: a source, a void, and

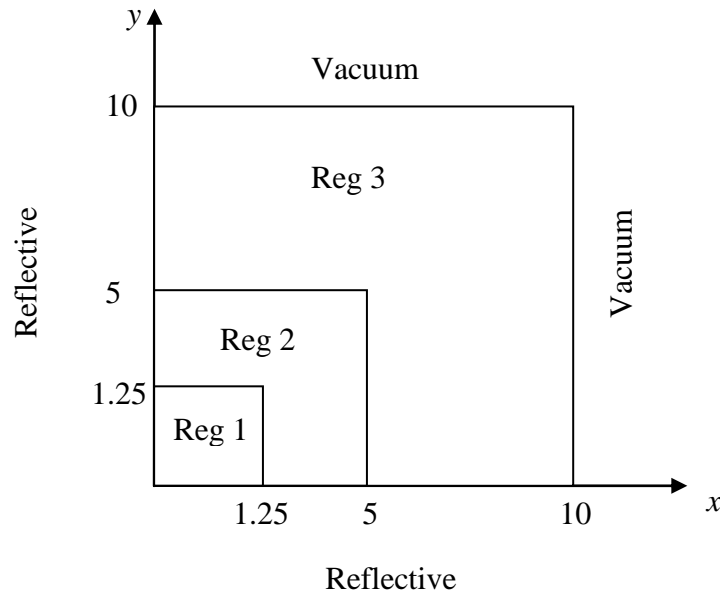


Figure 2.13. Geometry of Square Source Void Problem

an absorber. The boundary conditions are assumed to be reflective on the left and lower boundaries and vacuum on the top and right boundaries.

Table 2.8. Cross Section Data for Square Source Void Problem

Region	Q	σ_t	σ_s
1	6.4	0.2	0.19
2	0	0	0
3	0	0.2	0.19

Figures 2.14 and 2.15 show the results of each method as a two dimensional plane and Figure 2.16 shows the results along $x = 5.625$. As before, the size of the matrix for the discontinuous method is $4*M*N$ where the size of the matrix for the LS method is only $M*N$.

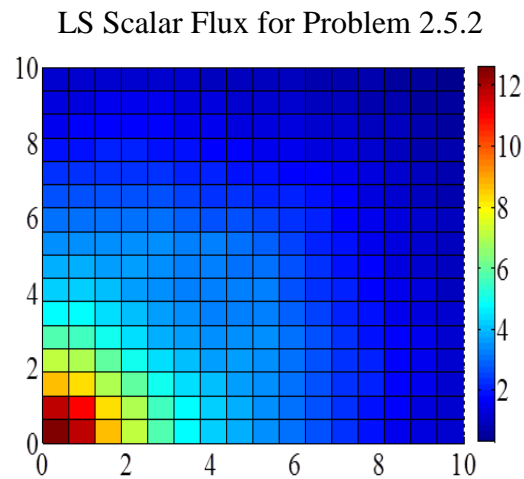


Figure 2.14. Scalar Flux for the Square Source Void Problem Using the LSFEM

DFEM Scalar Flux for Problem 2.5.2

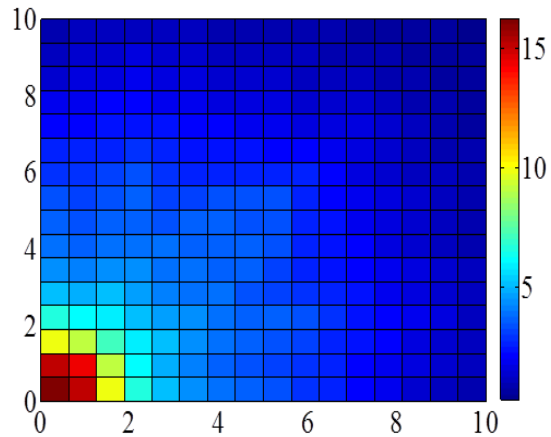


Figure 2.15. Scalar Flux for the Square Source Void Problem Using DFEM

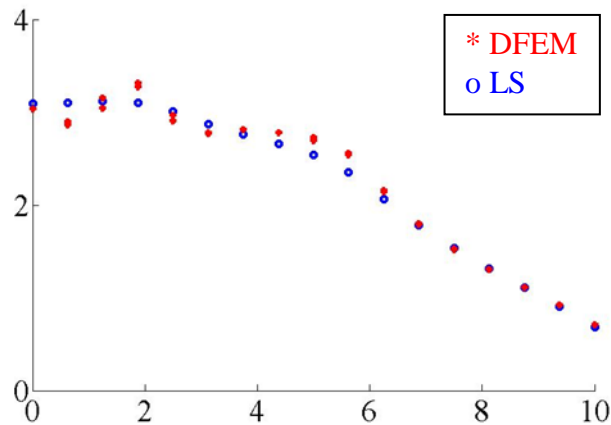
Scalar Flux along $x = 5.625$ for Problem 2.5.2

Figure 2.16. Resulting Flux Along Line $x = 5.625$ for Square Source Void Problem

The two dimensional problem is only tested with respect to changes in the number of angles and not in step size. This is to accommodate the memory restrictions on using a single machine. The tests are performed at three different angular quadratures called symmetric level sets which determine a certain number of angles around the unit circle.

Tables 2.9 and 2.10 show the comparison between the CG and BICGSTAB methods with the physical based preconditioners, the physical based incomplete Cholesky for the CG method and the incomplete LU for the BICGSTAB method. The improvement in run time in each case is not as dramatic as with the one dimensional problem, but we still see improvements of about 5 to 20 times.

Table 2.9. Results of Preconditioning on the LSFEM for Square Source Void Problem

Method	Angular Quadratures for 16x16 Mesh		
	S8	S10	S12
CG	2.28 s, 511 it.	5.53 s, 586 it.	11.5 s, 648 it.
PCG - PB IC	0.491 s, 98 it.	1.06 s, 104 it.	2.02 s, 106 it.

Table 2.10. Results of Preconditioning on the DFEM for the Square Source Void Problem

Method	Angular Quadratures for 16x16 Mesh		
	S8	S10	S12
BICGSTAB	5.16 s, 280 it.	11.3 s, 291 it.	24.3 s, 319 it.
PBICGSTAB - PB IC	0.382 s, 17 it.	0.682 s, 15 it.	1.28 s, 15 it.

Tables 2.11 and 2.12 show the comparison of the physical based preconditioner with the other black box and direct preconditioners. In these cases the physical based preconditioners actually outperform Matlab's incomplete LU factorizations. It is also interesting to note that the direct Cholesky factorization and LU factorization outperform the incomplete factorizations. This may be due to the two dimensional problem having a

more complex structure than the one dimensional problem making the incomplete factorization less effective.

Table 2.11. Comparison of Preconditioners on LSFEM for Square Source Void Problem

PCG Method	Angular Quadratures for 16x16 Mesh		
	S8	S10	S12
None - CG	2.28 s, 511 it.	5.53 s, 586 it.	11.5 s, 648 it.
Jacobi (Diagonal)	1.43 s, 286 it.	3.45 s, 305 it.	6.56 s, 319 it.
SSOR	1.22 s, 113 it.	2.85 s, 125 it.	5.68 s, 134 it.
None - DC	0.994 s	2.44 s	5.17 s
IC	0.995 s, 96 it.	2.10 s, 99 it.	4.08 s, 103 it.
Physical Based - IC	0.491 s, 98 it.	1.06 s, 104 it.	2.02 s, 106 it.
Physical Based - DC	0.143 s, 16 it.	0.281 s, 17 it.	0.491 s, 17 it.

Table 2.12. Comparison of Preconditioners on DFEM for Square Source Void Problem

PCG Method	Mesh Size (n -spatial, m -direction)		
	S8	S10	S12
None - BICGSTAB	5.16 s, 280 it.	11.3 s, 291 it.	24.3 s, 319 it.
Jacobi (Diagonal)	4.04 s, 200 it.	8.18 s, 198 it.	15.8 s, 197 it.
None - DLU	3.60 s	10.9 s	23.3 s
Physical Based - DLU	0.980 s, 5 it.	1.55 s, 5 it.	2.32 s, 5 it.
ILU	0.677 s, 14 it.	1.33 s, 13 it.	2.49 s, 4 it.
Physical Based - ILU	0.382 s, 17 it.	0.682 s, 15 it.	1.28 s, 15 it.

2.5.3 Scattering Ratio

The final problem is to test the effectiveness of the preconditioners as the degree of scattering is increased. This problem will have the same geometric structure as the square source void problem with the slight difference that we will allow the scattering cross-section σ_s value to vary between 0 and 0.19 within Region 1 and Region 3. These results are only tested for the LSFEM at the S8 angular quadrature. The results are shown in Table 2.13. Figure 2.17 shows the scalar flux along the line $x = 5.625$ for three of the scattering cross-sections.

Table 2.13. Comparison of Preconditioners for LSFEM with varying Scattering Values

PCG Method	16x16 Mesh, S8				
	$\sigma_s = 0.01$	$\sigma_s = 0.05$	$\sigma_s = 0.1$	$\sigma_s = 0.15$	$\sigma_s = 0.19$
None - CG	2.11 s, 456 it.	2.16 s, 466 it.	2.15 s, 475 it.	2.27 s, 493 it.	2.28 s, 511 it.
Jacobi (Diagonal)	1.25 s, 257 it.	1.33 s, 262 it.	1.30 s, 267 it.	1.32 s, 278 it.	1.43 s, 286 it.
SSOR	1.09 s, 102 it.	1.10 s, 104 it.	1.12 s, 106 it.	1.13 s, 107 it.	1.22 s, 113 it.
None - DC	0.910 s	0.848 s	0.844 s	0.899 s	0.994 s
IC	0.882 s, 81 it.	0.885 s, 82 it.	0.872 s, 84 it.	0.944 s, 88 it.	0.995 s, 96 it.
Physical Based - IC	0.454 s, 87 it.	0.481 s, 89 it.	0.478 s, 91 it.	0.479 s, 93 it.	0.491 s, 98 it.
Physical Based - DC	0.0463 s, 257 it.	0.070 s, 7 it.	0.0978 s, 10 it.	0.116 s, 13 it.	0.143 s, 16 it.

The main thing of interest here is the performance of the preconditioner as the scattering ratio increases. All of the preconditioners vary with respect to the scattering

ratio, some more dramatically than others. The main preconditioner of interest is the physical based direct cholesky preconditioner which varies from 45 times improvement to only a 15 times improvement. The incomplete cholesky preconditioner varies from 2.4 times improvement to a 2.3 times improvement. These are not exhaustive results and should not be treated as such, but it seems that the physically based direct Cholesky is much more affected by the change in scattering ratio than the other preconditioners.

Scalar Flux along $x = 5.625$ for Problem 2.5.2

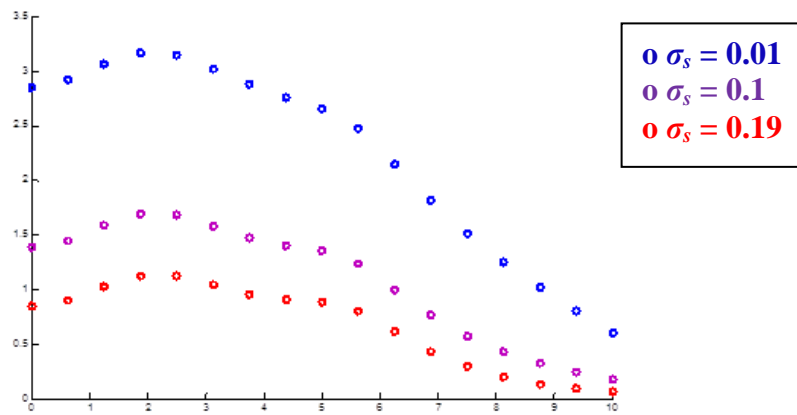


Figure 2.17. Flux Along Line $x = 5.625$ for Several Scattering (σ_s) Values

2.6 Conclusions and Future Work

As can be seen in the results, dramatic improvements can be made on the run times for linear systems that result from finite element approximations of the first order formulation of the transport equation. The physically based preconditioner originally proposed in [41] on the second order formulation has been shown to be effective on the first order formulation of the transport equation as well. Some of these results were presented in [44] as well. The physically based preconditioner was shown to be more

effected by changes in the scattering ration than other preconditioners, but still showed much improvement to the other preconditioners.

This project was not an exhaustive result of all preconditioners, linear solution methods, or even differential equation solution methods and further results can be done in each area. Other finite element methods that were tried were a discontinuous least squares finite element method (DLSFEM) and a mixed finite element method (MFEM). The DLSFEM broke down in voids and alternative methods for improvement were out of the scope of this work so it was dropped in favor of the DFEM and LSFEM. The MFEM as mentioned above showed less stability compared to the DFEM and LSFEM and was similarly dropped. Both of these methods could be explored further especially the LSFEM and the physical based preconditioner could be coupled with the MFEM on second order formulation of the transport problem as well.

Other linear solution methods that could be compared are generalized minimum residual methods and more especially multigrid methods, which, although complex, have been shown to be the fastest linear solvers [2]. As seen above there are also many ways of implementing the preconditioners which could be explored further. Further work could also be done on larger parallel computing architectures to see the results of each method there. This will be explored in some detail in chapter 4, but only as it relates to parallel computing on graphics cards using CUDA.

In summary, the physical based preconditioning proved effective on first order formulations of the transport equation and further studies could be done to determine better differential equation solution methods, linear system solution methods, or

preconditioners and preconditioner methods. In the next chapter we will see the results of a physically based preconditioner on fluid transport problem.

CHAPTER 3

**PHYSICALLY BASED PRECONDITIONING FOR THE MIXED FINITE
ELEMENT METHOD APPLIED TO A HOMOGENIZED FORM OF THE FLOW
EQUATION IN POROUS MEDIA**

The second major section of this research study was based on preconditioning the linear systems resulting from mixed finite element methods (MFEMs) applied to a flow equation for porous media. This was a continuation of previous research conducted by Koebbe in [11]. In [11] a modification of the MFEM was used to increase the efficiency of the conjugate gradient (CG) method for the linear system resulting from the MFEM for homogenized forms of the flow equation. To further increase the efficiency of the CG solver, in this study a physically based preconditioner was applied to the linear solver.

The mixed finite element method has been studied for some time and was originally proposed in [45]. Consider the simple flow equation

$$\nabla \cdot (T\nabla h) = q \quad (3.1)$$

in one and two dimensions, where T is a matrix in $R^{2 \times 2}$ or $R^{3 \times 3}$, h is a scalar function of two or three variables, and q is a scalar function of two or three variables. The basic idea of the method is to split the above second order equation into a system of two first order equations given by

$$v = T\nabla h \quad (3.2)$$

$$\nabla \cdot v = q \quad (3.3)$$

where v is a vector in R^2 or R^3 . The term mixed comes from the determination of the basis functions within the finite element method. The basis functions for the velocity are often

chosen to be a higher order than the order of the basis functions for the head or pressure variable; for examples, see [9], [11], [45], and [46].

So, in other words we divide the original equation into a two equations related to the velocity and apply the finite element method to both simultaneously. This allows for the velocity v and the head or pressure variable h to be solved simultaneously. This makes it possible to solve for the velocity more accurately [11]. However, this also greatly increases the size of the linear system resulting from the finite element method which can decrease efficiency. In some cases this greater size can be simply reduced to a smaller size, but in other cases requires more effort. We will discuss a couple cases below.

In [9] various aspects of the MFEM were studied for the case when the transmissivity tensor T is a diagonal matrix of the form

$$T = I\omega(x, y) \quad (3.4)$$

where I is the identity matrix and ω is a function of two variables. In [9] is also included an analytic solution of the flow equation for the identity transmissivity. The analytic solution was used to test the validity of the mixed finite element codes written for this study. When a diagonal transmissivity is used the linear system resulting from the MFEM has a structure that is easily simplified to reduce the size of the system. This is shown below in Equation (3.57). This is not true, however, for the case when T is a full matrix.

When T is a full matrix, solving the above equation becomes much more difficult. The resulting linear system becomes a full system like in Equation (3.32) and is not as easily reduced to a simpler system like in Equation (3.57). However, there are still some things that can be done to simplify the system, but first we will consider what cases might include a full transmissivity tensor.

For many problems the transmissivity is diagonal where only the transmissivity in the coordinate directions is specified and the correlated transmissivities are zero (for example, see [10]). One case where we encounter full transmissivity tensors is with porous media. Various methods have been used for modeling fluid flow within porous media. Examples of continuous and discontinuous methods can be found in [47] and [48] respectively. Examples of porous media include water or oil in underground reservoirs. These "reservoirs" are actually water or oil mixed with other sediments, and wells or pumps extract the water or oil from the other sediments. Porous media problems are often associated with the method of homogenization, see [10], [49], and [50]. Homogenization could be called an averaging procedure for the transmissivity of porous media. For many problems, like the example in Equation (3.4), the transmissivity is variable with respect to the domain. Homogenization is a method of averaging the variable transmissivity to a single constant transmissivity, greatly simplifying the problem. Thus the method of homogenization creates a new problem whose solution approximates the solution of the original problem. The accuracy of the approximation depends on the problem itself. This particular study does not deal with the accuracy of homogenization, but it has been shown to work well for media with a periodic porosity structure; see [10], [49], [51]. Some details on the homogenization procedure will be given below. Further details can be found in [49], [50], [10], and [51].

Sometimes when using homogenization in porous media problems, a flow equation that contains only diagonal transmissivity tensors can result in a full tensor. The fluid flow problem studied in [11], which this research continues, is such a problem. In [11] a modification of the mixed finite element method in [9] is used to simplify the case when

the transmissivity matrix is full. This is a projection method and details are given below in Section 3.4. The study of this paper was to take the linear system resulting from the modified mixed finite element method applied to the homogenized flow equation and see the effects of physically based preconditioning on the conjugate gradient method for the linear system. This began by developing mixed finite element codes in two and three dimensions. Linear solvers and preconditioners were then written to solve the resulting linear system. An explanation of the codes can be found in the appendix. Codes were also written for homogenization of the porous media, but they are not shown here. Details can be found in [52]. A linear solver was also written for parallel processing on graphics processing units using CUDA. Details of this are given in Chapter 4.

To summarize, there are four levels of improvement used to increase the efficiency of solving the flow problem, Equation (3.1), using the mixed finite element method. The first and second have been done previously which are the homogenization of the transmissivity tensor and the projection method to modify the mixed finite element method. Two more levels of improvement are applied here, preconditioning of the linear system resulting from the mixed finite element method and preconditioning of the conjugate gradient solver in parallel in CUDA.

The remainder of the chapter is organized as follows. There will first be a brief background on fluid flow in porous media. This will be followed by the background and implementation of each of the four methods of improvement. This will be followed by numerical results and discussion. Three problems will be studied. The first will be a source sink problem, sometimes called the quarter five spot problem; see [53] and [54] for examples. The second will be a one dimensional flow problem, and the third will be a

varied transmissivity problem. Good improvement on the efficiency of the CG solver for the physically based preconditioner are seen in each case. The physically based preconditioner is also compared with other black box preconditioners. The results will be followed by the conclusion and possible future work.

3.1 Fluid Flow in Porous Media

It is common in engineering and scientific problems to have to deal with materials formed from multiple constituents [49]. One example is modeling oil extraction from underground reservoirs [50]. In general, when modeling fluid flow, one would hope for the simplest case, where the fluid is contained in a single open space of normal size (ie cubic, spherical, etc.), and that the only thing within the open space is the fluid being extracted. Then the corresponding model of flow would be an exercise of basic calculus. However, within these reservoirs, the fluid is generally contained in regions of varying porosity or permeability (ie the ability of the oil to flow freely). So in some places, the fluid will flow relatively freely and in others flow relatively slowly. An example of such a structure is given in Figure 3.1.

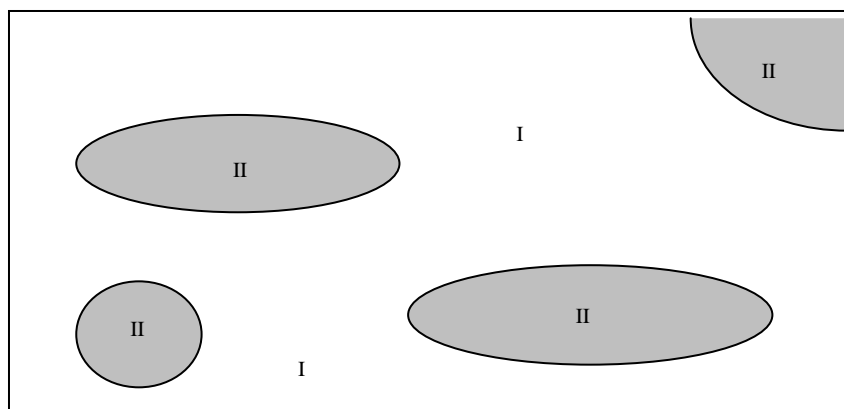


Figure 3.1. Depiction of Porous Media

These changes in transmissivity generally occur on a very small scale compared with the entire reservoir. To model the fluid flow accurately, these changes need to be accounted for. Another important factor is that the region where the fluid is actually being extracted is small compared to the size of the actual reservoir; for example the pipe water gets pumped through from an underground reservoir is relatively small compared to the size of the reservoir itself. This difference in scale coupled with the small scale of the changes in permeability can require the computational grid of a numerical method to be very fine over a large region, which greatly increases the complexity of the problem. Because of this it becomes very important to find ways to improve the efficiency of methods for solving flow problems in porous media. We will look at four ways of improving that efficiency.

3. 2 MFEM Approximation

We will start by looking at the flow equation and then show some of the details of the implementation of each of the improvements: homogenization, projection of the mixed finite element method, physically based preconditioning, and parallel computing. The simple flow equation is given by

$$\nabla \cdot (T\nabla h) = q \quad (3.5)$$

in two and three dimensions, where T is a matrix in $R^{2 \times 2}$ or $R^{3 \times 3}$, h is a scalar function of two or three variables, and q is a scalar function of two or three variables. For these problems we will assume that T is periodic on a small scale ε . The boundary condition for this problem will be as follows.

$$\nabla h \cdot \nu = 0 \quad (3.6)$$

where v is the unit normal vector.

3.2.1 Homogenization

The method of homogenization has become a classical method in a variety of fields including asymptotic analysis, composite media theory, wave propagation, effective media theory, bulk property theory, and others [49] [55]. The method goes by various names, the most general of which is perhaps the method of multiple scales, or more specifically the method of two scales. The theory for composite media has been studied extensively for more than 100 years, with, as Milton puts it “an explosion of ideas in the last four decades” [55]. The literature on homogenization is quite extensive. A good summary can be found in [51].

As explained above the method of homogenization is a sort of averaging procedure. A common calculus problem is to compute the work needed to pump a certain volume of water a certain height out of its container. This problem reduces to a simple integral. One of the assumptions of the problem is that the water is in a homogeneous state (i.e. the water isn't mixed with anything else). If we were to say that the water is mixed into sand, then the problem becomes much more complicated. By throwing rocks, debris, geological layers, and so on the problem gets pretty complex pretty quickly. One thing to note, however, in the water and sand example is that, if the sand is pretty homogeneous as well, then it would probably be safe to assume that the water and sand mixture is relatively homogeneous, which means that the water would flow through the sand at the same rate regardless of where the water is in the sand. In terms of Equation 1 this means that although the transmissivity T will change by adding sand to the water, the

transmissivity will nevertheless remain relatively constant throughout the domain. This is the idea behind homogenization.

In other words, it takes into account the small scale permeabilities over the entire reservoir, but also allows for a coarser computational grid, thus decreasing the number of computations. It does this by averaging the permeability over the entire region. A couple simple examples are shown in Figures 3.2 and 3.3 as found in [10] and [50] respectively.

The transmissivity tensors in each region of Figure 3.2 as well as the corresponding homogenized tensor are given by

$$T_I = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, \quad T_{II} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad T_H = \begin{bmatrix} 6.52 & 0 \\ 0 & 6.52 \end{bmatrix} \quad (3.7)$$

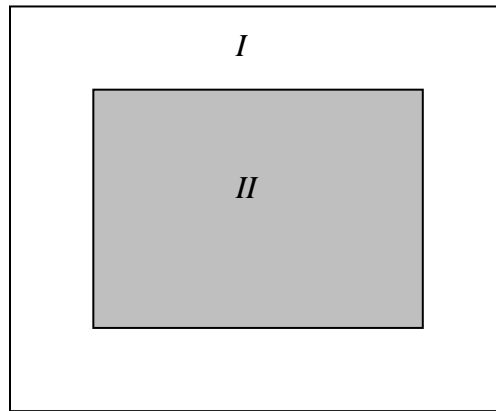


Figure 3.2 Example of Periodic Two Phase Flow Structure

The transmissivity tensors in each region of Figure 3.3 as well as the corresponding homogenized tensor are given by

$$T_I = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, \quad T_{II} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad T_H = \begin{bmatrix} 5.5 & 0 \\ 0 & 1.81 \end{bmatrix} \quad (3.8)$$

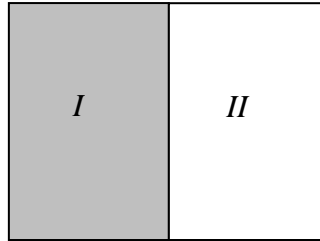


Figure 3.3 Linear Example of Two Phase Flow Structure

The term averaging is used loosely above. However it is related but not limited to the standard averages like the arithmetic, geometric, or harmonic averages. Note that 5.5 is the arithmetic average of the T_I value and the T_{II} value in Figure 3.3 and that 1.81 is the harmonic average.

Using this averaging technique produces a permeability that is either constant over the computational domain of the problem or at least one that does not change as rapidly that can be computed on a coarser scale. For the problems above, since the the two original transmissivities given were constant, then the homogenized transmissivity is also constant, but this is not always the case. The generalized procedure for computing the homogenized tensor is outlined below.

3.2.1.1 Homogenization Implementation

Much of the analysis presented here can also be found in [10] and [50]. We will first go through the method of multiple scales and find equations for the function h on the first order. Next, we will find the homogenized or average equation. We will then go over the intermediate step of finding the homogenized coefficient that goes into the homogenized equation.

We start with the coupled velocity form of the flow equation

$$v = T\nabla h \quad (3.2)$$

$$\nabla \cdot v = q \quad (3.3)$$

We then proceed with the method of multiple scales.

$$T = T(y), T(y + \varepsilon) = T(y) \text{ for some period } \varepsilon \quad (3.9)$$

$$h = h(x,y), h = h_0 + \varepsilon h_1 + \dots \quad (3.10)$$

$$v = v(x,y), v = v_0 + \varepsilon v_1 + \dots \quad (3.11)$$

$$\nabla_{x'} = \nabla_x + \frac{1}{\varepsilon} \nabla_y \quad (3.12)$$

where $x = x'$ and $y = x'/\varepsilon$, where x' is the variable of the original equation (i.e. $h = h(x')$).

With these assumptions we have the following equations at the first and second orders

ε^{-1} :

$$0 = -T\nabla_y h_0 \quad (3.13)$$

$$-\nabla_y \cdot v_0 = 0 \quad (3.14)$$

ε^0 :

$$v_0 = -T(\nabla_x h_0 + \nabla_y h_1) \quad (3.15)$$

$$-(\nabla_x \cdot v_0 + \nabla_y \cdot v_1) = q \quad (3.16)$$

At the first order, the Equations (3.13) and (3.14) imply that $h_0 = h_0(x)$, $v_0 = v_0(x)$ so long

as T is positive definite. We further assume that $h_1 = \omega(y) \cdot \nabla_x u_h$, where

$\omega(y) = [w_1(y) \ w_2(y)]^T$. This seems intuitive since the expansion on h can be compared

to a Taylor Series expansion. With the first assumption Equation (3.15) becomes

$$v_0 = -T(\nabla_x h_0 + [\nabla_y w_1 \ \nabla_y w_2] \nabla_x h_0) = -T(I + [\nabla_y w_1 \ \nabla_y w_2]) \nabla_x h_0 \quad (3.17)$$

Equation (3.14) then implies that

$$\begin{aligned}
0 &= -\nabla_y \cdot v_0 = \nabla_y \cdot T(I + [\nabla_y w_1 \quad \nabla_y w_2]) \nabla_x h_0 \\
&= \nabla_y \cdot \left((Te_1 + T\nabla_y w_1) \frac{\partial h_0}{\partial x_1} + (Te_2 + T\nabla_y w_2) \frac{\partial h_0}{\partial x_2} \right)
\end{aligned} \tag{3.18}$$

where e_1 and e_2 are the column vectors of the 2x2 identity matrix. Assuming that $\nabla_x h_0 \neq [0 \quad 0]^T$ we have the following equations

$$\nabla_y \cdot (T\nabla_y w_1) = -\nabla_y \cdot Te_1 \tag{3.19}$$

$$\nabla_y \cdot (T\nabla_y w_2) = -\nabla_y \cdot Te_2 \tag{3.20}$$

Equations (3.19) and (3.20) are referred to as the local problem since they are solved on the small scale or in the fast variable y . Once w_1 and w_2 are found, we return to Equation (3.17). If we assume that the second term in the expansion of the velocity is periodic (ie v_1 is periodic in the fast scale y) and integrate each side of Equation (3.16) over the fast variable y we have

$$q^\# = \nabla_x \cdot (T^\# \nabla_x h_0) \tag{3.21}$$

$$T^\# = \frac{1}{|Y|} \int_Y T(I + [\nabla_y w_1 \quad \nabla_y w_2]) dY \tag{3.22}$$

$$q^\# = \frac{1}{|Y|} \int_Y q dY \tag{3.23}$$

For our experiments, the function q is assumed to be constant over the fast variable so that $q^\# = q$, but in general, if q does depend on the fast variable then this integral can be estimated numerically as well. Equation (3.21) can then be solved using a standard finite element method to find a first order solution. Specifically a mixed finite element method is used.

3.2.2 Projected Mixed Finite Element Method

The specific model problem for the fluid transport code is the homogenized simple flow Equation (3.21). For sake of simplicity we will revert back the original form of Equation (3.5) and assume that it has already been through the homogenization process.

$$\nabla \cdot (T\nabla h) = q \quad (3.5)$$

where h is the head or pressure variable, T is a transmissivity tensor, and q is the product of the storativity or porosity times the change in pressure with respect to time. The boundary condition is the same as in (3.6). There are computational difficulties associated with this problem as well. One difficulty that arises is when the transmissivity T is a full matrix. In this case, the linear system matrix resulting from the mixed finite element method is full as well, greatly increasing the computational difficulty. This problem was originally addressed in [11] where a projection method was devised to speed up the linear solver to obtain a solution. In this work, we extend the work of [11] and apply preconditioning to the projected form of the full transmissivity tensor case. We will first go through the implementation of the mixed finite element method and the projection method. We will then give more detail on the diagonal transmissivity preconditioner and look at the results of the preconditioning.

We start with the velocity form of the flow Equations (3.2) and (3.3)

$$v = T\nabla h \quad (3.2)$$

$$\nabla \cdot v = q \quad (3.3)$$

The mixed finite element method proceeds in a similar fashion to standard finite element methods. The pressure variable h will be assumed to be a linear combination of piecewise

constant basis functions and the velocity variable v will be assumed to be a linear combination of piecewise linear basis functions. Specifically

$$\begin{aligned} v &= \sum_j \sum_i \left(v_{ij}^x \phi_i(x) \chi_j(y) \bar{e}_1 + v_{ij}^y \chi_i(x) \phi_j(y) \bar{e}_2 \right) \\ h &= \sum_j \sum_i h_{ij} \chi_i(x) \chi_j(y) \end{aligned} \quad (3.24)$$

where ϕ is defined as

$$\phi_i(x) = \begin{cases} \frac{x - x_i}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \quad (3.25)$$

and χ is defined as

$$\chi_i(x) = \begin{cases} 1 & x \in [x_{i-1}, x_i] \\ 0 & \text{otherwise} \end{cases} \quad (3.26)$$

The finite element method then proceeds in a usual way according to a Galerkin method.

A test function from each of the two function spaces is multiplied by each equation above, and each equation is then integrated over the domain.

$$\int_A \left((T^{-1}v) \cdot u - \nabla h \cdot u \right) dA = 0 \quad (3.27)$$

$$\int_A (\nabla \cdot v) w dA = \int_A q w dA \quad (3.28)$$

After integration by parts and applying the boundary condition $n \cdot v = 0$ where n is the outward unit normal vector we have

$$\int_A \left((T^{-1}v) \cdot u + h \cdot \nabla u \right) dA = 0 \quad (3.29)$$

$$\int_A (\nabla \cdot v) w dA = \int_A q w dA \quad (3.30)$$

which in its discretized form gives the linear system

$$\begin{aligned} \sum_j \sum_i \int_A \left((T^{-1} (v_{ij}^x \phi_i(x) \chi_j(y) \bar{e}_1 + v_{ij}^y \chi_i(x) \phi_j(y) \bar{e}_2)) \cdot u + h_{ij} \chi_i(x) \chi_j(y) \cdot \nabla u \right) dA &= 0 \\ \sum_j \sum_i \int_A \left(\nabla \cdot (v_{ij}^x \phi_i(x) \chi_j(y) \bar{e}_1 + v_{ij}^y \chi_i(x) \phi_j(y) \bar{e}_2) \right) w dA &= \int_A q w dA \end{aligned} \quad (3.31)$$

The linear system has the following matrix form.

$$\begin{bmatrix} M_{xx} & M_{xy} & N_1 \\ M_{yx} & M_{yy} & N_2 \\ N_1 & N_2 & D \end{bmatrix} \begin{bmatrix} v^x \\ v^y \\ h \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_h \end{bmatrix} \quad (3.32)$$

where M_{xx} and M_{yy} are symmetric tridiagonal matrices representing the transmissivity in each coordinate direction, M_{xy} and M_{yx} are sparse matrices representing the transmissivity in the off-coordinate directions, N_1 and N_2 represent differencing matrices, D contains the time-dependent information as well as any initial conditions on the pressure, v^x and v^y are the vector forms of the velocity, h is now the vector form of the pressure, and r_x , r_y , r_h are the right hand sides for the velocity and pressure respectively and contain the boundary information. This form is symmetric and positive definite lending itself to efficient iterative solvers like the conjugate gradient method. If the transmissivity tensor T is a full matrix, then the matrix above will be full as well. This form of the equation is further modified using a projection method.

3.2.2.1 Projection

The projection method, the second level of improvement, which can also be found in [11], starts with the velocity form of the flow Equation (3.2) and (3.3)

$$v = T\nabla h \quad (3.2)$$

$$\nabla \cdot v = q \quad (3.3)$$

We separate the diagonal velocity from the off-diagonal velocity according to the transmissivity tensor and rewrite Equations (3.2) and (3.3). In two dimensions we have

$$T = T_d + T_n \quad (3.33)$$

$$v_d = T_d \nabla h \quad (3.34)$$

$$\nabla \cdot v_d + \nabla \cdot (T_n \nabla h) = q \quad (3.35)$$

where the off-diagonal term in Equation (3.35) can be rewritten in terms of the diagonal velocity as follows.

$$(T_n \nabla h) = \begin{pmatrix} 0 & T_{xy} \\ T_{yx} & 0 \end{pmatrix} \begin{pmatrix} \frac{\partial h}{\partial x} \\ \frac{\partial h}{\partial y} \end{pmatrix} = \begin{pmatrix} T_{xy} \frac{\partial h}{\partial y} \\ T_{yx} \frac{\partial h}{\partial x} \end{pmatrix} = K v_d \quad (3.36)$$

where

$$K = \begin{pmatrix} 0 & \frac{T_{xy}}{T_{yy}} \\ \frac{T_{yx}}{T_{xx}} & 0 \end{pmatrix} \quad (3.37)$$

and thus Equation (35) becomes

$$\nabla \cdot v_d + \nabla \cdot (K v_d) = q \quad (3.38)$$

Proceeding with the Galerkin Method, similar to Equation (3.38) we would have

$$\int_A (\nabla \cdot v_d + \nabla \cdot (K v_d)) w dA = \int_A q w dA \quad (3.39)$$

We note here that the second term Kv_d is not in the same trial functions space as v_d [11].

For this reason, a projection of Kv_d onto the same trial function space as v_d is done. We

thus end up with a coupled system of three equations. In the variational form we have

$$\int_A \left((T^{-1}v_d) \cdot u + h \cdot \nabla u \right) dA = 0 \quad (3.40)$$

$$\int_A (\nabla \cdot v_d + \nabla \cdot F) w dA = \int_A q w dA \quad (3.41)$$

$$\int_A (F \cdot \mu) dA = \int_A ((Kv_d) \cdot \mu) dA \quad (3.42)$$

where u and μ are test functions from the velocity trial space and w is a test function from the head or pressure trial space. This system can also be discretized similar to Equation

(3.31) above. The resulting linear system in three dimensions has the form

$$\begin{bmatrix} M_{xx} & 0 & 0 & N_1 & 0 & 0 & 0 \\ 0 & M_{yy} & 0 & N_2 & 0 & 0 & 0 \\ 0 & 0 & M_{zz} & N_3 & 0 & 0 & 0 \\ N_1^T & N_2^T & N_3^T & D & N_1^T & N_2^T & N_3^T \\ 0 & -B_{xy} & -B_{xz} & 0 & A_{xx} & 0 & 0 \\ -B_{yx} & 0 & -B_{yz} & 0 & 0 & A_{yy} & 0 \\ -B_{zx} & -B_{zy} & 0 & 0 & 0 & 0 & A_{zz} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ h \\ f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ r_h \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.43)$$

where A_{xx} , A_{yy} , and A_{zz} represent velocities in the coordinate directions and the B matrices represent the projection between the original velocity space and the velocity space of Kv_d .

The A 's have the same structure as the diagonal velocity M matrices, the B 's have the same structure as the off-diagonal transmissivity matrices, and the f 's represent velocity trial functions for the projection equation.

Looking at the bottom three block equations we have

$$f_x = A_{xx}^{-1} B_{xy} v_y + A_{xx}^{-1} B_{xz} v_z \quad (3.44)$$

$$f_y = A_{yy}^{-1} B_{yx} v_x + A_{yy}^{-1} B_{yz} v_z \quad (3.45)$$

$$f_z = A_{zz}^{-1} B_{zx} v_x + A_{zz}^{-1} B_{zy} v_y \quad (3.46)$$

Applying these to the linear system (3.43) we can rewrite the system in a reduced form.

$$(3.47) \quad \begin{bmatrix} M_{xx} & 0 & 0 & N_1 \\ 0 & M_{yy} & 0 & N_2 \\ 0 & 0 & M_{zz} & N_3 \\ N_1^T + N_2^T A_{yy}^{-1} B_{yx} + N_3^T A_{zz}^{-1} B_{zx} & N_2^T + N_1^T A_{xx}^{-1} B_{xy} + N_3^T A_{zz}^{-1} B_{zy} & N_3^T + N_1^T A_{xx}^{-1} B_{xz} + N_2^T A_{yy}^{-1} B_{yz} & D \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ h \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ r_h \end{bmatrix}$$

Block row-reducing this can be further simplified to the form

$$(3.48) \quad \begin{bmatrix} M_{xx} & 0 & 0 & N_1 \\ 0 & M_{yy} & 0 & N_2 \\ 0 & 0 & M_{zz} & N_3 \\ 0 & 0 & 0 & A \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ h \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ R \end{bmatrix}$$

where

$$(3.49) \quad A = D - \left(N_1^T + N_2^T A_{yy}^{-1} B_{yx} + N_3^T A_{zz}^{-1} B_{zx} \right) M_{xx}^{-1} N_1 - \left(N_2^T + N_1^T A_{xx}^{-1} B_{xy} + N_3^T A_{zz}^{-1} B_{zy} \right) M_{yy}^{-1} N_2 - \left(N_3^T + N_1^T A_{xx}^{-1} B_{xz} + N_2^T A_{yy}^{-1} B_{yz} \right) M_{zz}^{-1} N_3$$

$$(3.50) \quad R = r_h - \left(N_1^T + N_2^T A_{yy}^{-1} B_{yx} + N_3^T A_{zz}^{-1} B_{zx} \right) M_{xx}^{-1} r_x - \left(N_2^T + N_1^T A_{xx}^{-1} B_{xy} + N_3^T A_{zz}^{-1} B_{zy} \right) M_{yy}^{-1} r_y - \left(N_3^T + N_1^T A_{xx}^{-1} B_{xz} + N_2^T A_{yy}^{-1} B_{yz} \right) M_{zz}^{-1} r_z$$

So the reduced system

$$(3.51) \quad Ah = R$$

can be solved for the head variable h and then v_x , v_y , and v_z can be found by

$$(3.52) \quad v_x = M_{xx}^{-1} (r_x - N_1 h)$$

$$(3.53) \quad v_y = M_{yy}^{-1} (r_y - N_2 h)$$

$$(3.54) \quad v_z = M_{zz}^{-1} (r_z - N_3 h)$$

Thus, the projection method used above takes the original linear system, assuming the mesh sizes for v_x , v_y , and v_z are equal, and reduces it by almost a factor of four. The resulting matrix is still symmetric positive definite and the conjugate gradient method can still be applied. We will not here go over the conjugate gradient method or iterative solvers, but instead refer to that section in Chapter 2 for further information.

3.2.3 Physically Based Preconditioner

The third step in improving the efficiency of solving the flow equation is preconditioning the linear system within the conjugate gradient solver. For the algorithm and explanation of the preconditioned conjugate gradient method, see Chapter 2. The preconditioner that will be applied to system (51) is the diagonal transmissivity solution to the simple flow Equation (3.1). In other words the simple flow equation is solved with T being a diagonal matrix $T = T_d$. This results in a much sparser linear system

$$\begin{bmatrix} M_{xx} & 0 & 0 & N_1 \\ 0 & M_{yy} & 0 & N_2 \\ 0 & 0 & M_{zz} & N_3 \\ N_1^T & N_2^T & N_3^T & D \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ h \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ r_h \end{bmatrix} \quad (3.55)$$

which can be similarly reduced to the system

$$\begin{bmatrix} M_{xx} & 0 & 0 & N_1 \\ 0 & M_{yy} & 0 & N_2 \\ 0 & 0 & M_{zz} & N_3 \\ 0 & 0 & 0 & M \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ h \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ R \end{bmatrix} \quad (3.56)$$

where

$$M = D - N_1^T M_{xx}^{-1} N_1 - N_2^T M_{yy}^{-1} N_2 - N_3^T M_{zz}^{-1} N_3 \quad (3.57)$$

This system matrix M is much simpler and more sparse than the original system matrix (3.49) and the original linear system is modified using this matrix

$$M^{-1}Ax = M^{-1}b \quad (3.58)$$

The benefit of this preconditioning is in part shown in Figure 3.4 below. In this figure, a representation of the system matrix A is shown for a two dimensional problem with 32 spatial elements in each coordinate direction. The original matrix A , with a full transmissivity tensor, is a full matrix. With only the diagonal components of the transmissivity tensor, the resulting matrix M is block sparse with each of the off-diagonal blocks being diagonal. For a larger representation with larger blocks the sparsity would be much more dramatic than in the figure below.

$$A = \begin{bmatrix} x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x \end{bmatrix} \quad M = \begin{bmatrix} x & x & x & 0 & x & 0 & x & 0 \\ x & x & 0 & x & 0 & x & 0 & x \\ x & 0 & x & x & x & 0 & x & 0 \\ 0 & x & x & x & 0 & x & 0 & x \\ x & 0 & x & 0 & x & x & x & 0 \\ 0 & x & 0 & x & x & x & 0 & x \\ x & 0 & x & 0 & x & 0 & x & x \\ 0 & x & 0 & x & 0 & x & x & x \end{bmatrix}$$


$\text{cond}(A) = 45,010$

 $\text{cond}(M^{-1}A) = 132$

Figure 3.4. Example of Matrix Preconditioning for Fluid Transport

We can again see the dramatic effect that a preconditioner can have on the condition of a linear system. Since the preconditioner has significantly fewer non-zeros than the original matrix it requires much fewer computations to compute its inverse than the inverse of A . When applied within iterative methods like the preconditioned conjugate gradient, it adds relatively few computations to each iteration, but significantly reduces

the number of total iterations needed for convergence. In the results this physically based preconditioner is compared against other common preconditioners.

3.2.4 Parallel Implementation on GPU

The fourth level of improvement in efficiency of solving the simple flow equation is implementing the solution in parallel on graphics processing units using CUDA. The full details of using CUDA and the results from applying the linear solver in parallel are given in Chapter 4.

3.3 Numerical Results and Discussion

We will now look at the results from using the physically based preconditioner on three problems. The first two problems use a homogenized tensor which is explained in Section 3.7.1. The third lets the off-diagonal terms of the transmissivity tensor vary to see the effects of increasing the order of the corresponding off-block-diagonal elements on the preconditioner. The first two problems include a two and three dimensional source-sink problem, sometimes called the quarter 5 spot problem, and a two and three dimensional single phase flow problem. The third is the same as the second except that the transmissivity tensor is varied to see the effect on the preconditioner. The results are given below.

The tests were done in MATLAB using the built in sparse matrix structure within a preconditioned conjugate gradient method (PCG). The algorithm stops or converges when the tolerance (the squared residual norm) is 10^{-9} . Run times for all of the results were calculated using Matlab's tic and toc functions. The initial guess was set to be a

vector of all ones for the source-sink problem and set to zero for the other problems. CG methods were written with compressed sparse row (CSR) storage in Matlab and in C++. Only results using the standard Matlab storage are shown here to more easily compare the various preconditioners. The physically based preconditioner was tested against several built in and standard preconditioners similar to the preconditioners in Chapter 2 including incomplete Cholesky factorizations, successive over relaxation, and others. When the flow equation consists of a full tensor, the resulting matrix A as found in Equation (3.49) is a full matrix, so sparse storage is only used for the preconditioners and standard storage is used for the matrix A . We will first look at the source-sink problem with a homogenized domain, then at the single phase flow problem followed by the random transmissivity problem.

3.3.1 Homogenized Domain

In [11] a problem is presented where the permeability tensor is assumed to be periodic throughout the domain. This periodic pattern is shown below.

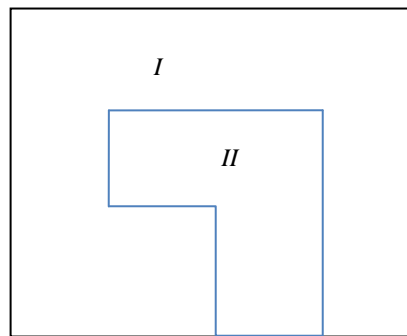


Figure 3.5. Diagram of Repeated Pattern for Homogenized Problem

In this case, given the two permeability tensors

$$T_I = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}, \quad T_H = \begin{bmatrix} 10.0 & 0.0 \\ 0.0 & 10.0 \end{bmatrix} \quad (3.59), (3.60)$$

and applying the method of homogenization we obtain the following full homogenized tensor

$$T_H = \begin{bmatrix} 1.49 & -0.08 \\ -0.08 & 1.89 \end{bmatrix} \quad (3.61)$$

This is the transmissivity used for the source-sink problem and the single phase flow problem.

3.3.2 Quarter 5-Spot Source-Sink Test Problem

The first problem is a two and three dimensional source sink problem given by Koebbe [11], and Wheeler and Ewing [9]. The right-hand side q of the simple flow Equation (3.1) is assumed to be a sum of dirac delta functions

$$q = \delta_{(0,0)}(x, y) - \delta_{(1,1)}(x, y) \quad (3.62)$$

The boundary condition is as explained above where the dot product of the velocity with the normal vector is assumed to be zero. In the case where $T = I$, an analytic solution can be found and is given in [9] in two dimensions as

$$h = \frac{1}{2\pi} \left(\log \left(\frac{\cosh(\pi(x+y)/2) + \cos(\pi(x-y)/2)}{\cosh(\pi(x+y)/2) - \cos(\pi(x-y)/2)} \right) + \sum_{n=1}^{\infty} (-1)^n \log \left(\frac{(\cosh(\pi(x+y-2n)/2) + \cos(\pi(x-y)/2))(\cosh(\pi(x+y+2n)/2) + \cos(\pi(x-y)/2))}{(\cosh(\pi(x+y-2n)/2) - \cos(\pi(x-y)/2))(\cosh(\pi(x+y+2n)/2) + \cos(\pi(x-y)/2))} \right) \right) \quad (3.63)$$

This solution was used to test the validity of the mixed finite element method. In Figure 3.6 the pressure is shown for the source-sink problem with the homogenized tensor and Tables 3.1 and 3.2 show the results of the preconditioning. The black box preconditioners tested

include successive over relaxation (SSOR), Jacobi (Diagonal), a block diagonal incomplete Cholesky (IC) factorization, an M-block incomplete Cholesky factorization, and an M-block direct cholesky factorization. The M-block preconditioners use a simplified block structure of the matrix A . Black box preconditioners are based solely on the matrix itself not on the problem that they are derived from, but the general block structure can be generally found by doing a couple simple searches on the matrix. The M-block here is a block diagonal with several off-diagonals. They are called M-block because they have the same matrix structure as the diagonal transmissivity preconditioner M , but the values are still taken from the original matix A . The physically based preconditioners include an incomplete cholesky factorization on the preconditioner M and a direct Cholsky factorization on the preconditioner M .

MFEM Pressure for Source Sink
Problem 3.3.2

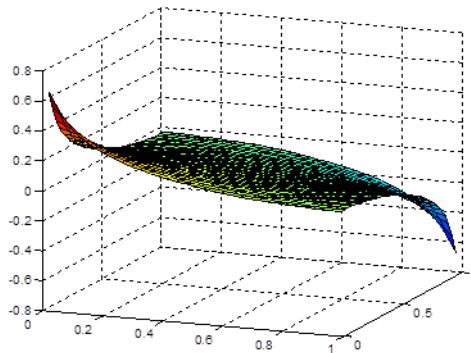


Figure 3.6. Resulting Pressure for MFEM Source Sink Problem with $m = n = 33$

The size of the resulting linear system is $m*n$ for two dimension and $m*n*o$ for three dimensions, where m , n , and o are the number of steps in the x , y , and z directions respectively. As mentioned above, for the full tensor case, Equation (3.61), the system

matrix is a full matrix. The mesh sizes were chosen for this research so that they can still be run on a single machine. The matrix sizes, therefore, were still small enough to

Table 3.1. Results of Preconditioning on MFEM 2D Source Sink Problem

PCG Method	Mesh Size (m -spatial in x , n -spatial in y)		
	$m = 45, n = 45$	$m = 65, n = 65$	$m = 85, n = 85$
SSOR	1.12 s, 119 it.	6.85 s, 172 it.	25.9 s, 225 it.
Jacobi (Diagonal)	0.894 s, 386 it.	5.51 s, 565 it.	21.4 s, 741 it.
None - CG	0.897 s, 396 it.	5.36 s, 575 it.	21.2 s, 752 it.
Block Diag - IC	0.745 s, 288 it.	4.35 s, 419 it.	16.7 s, 550 it.
None - DC	0.278 s	2.12 s	9.44 s
M-Block - IC	0.268 s, 100 it.	1.68 s, 145 it.	6.28 s, 191 it.
Physically Based - IC	0.266 s, 100 it.	1.64 s, 145 it.	5.89 s, 191 it.
M-Block - DC	0.128 s, 4 it.	0.609 s, 4 it.	1.49 s, 4 it.
Physically Based - DC	0.103 s, 3 it.	0.385 s, 3 it.	1.47 s, 3 it.

compare with direct methods, which can often be faster than iterative methods for smaller mesh sizes, especially optimized commercial methods like those in Matlab. Generally iterative methods are used solely for sparse matrices, and, as seen above, the direct Cholesky factorization performs better than the CG method and some of the preconditioners, but, even though the matrix is full, some of the preconditioners, including the physically based preconditioners still outperform the direct method. An alternate storage system for the matrix A can also be used which adds a few more calculations to each iteration of the CG method, but significantly reduces the amount

Table 3.2. Results of Preconditioning on MFEM 3D Source Sink Problem

Method	Mesh Size (m -spatial in x , n -spatial in y , o -spatial in z)	
	$m = 10, n = 10, o = 10$	$m=20,n=20,o=20$
Jacobi (Diagonal)	DNC	DNC
None - CG	0.132 s, 275 it.	18.1 s, 510 it.
None - DC	0.0338 s	9.78 s
Block Diag - IC	0.0558 s, 101 it.	5.17 s, 150 it.
SSOR	0.0512 s, 43 it.	4.47 s, 86 it.
Physically Based - IC	0.0210 s, 33 it.	2.86 s, 77 it.
M-Block Diag - IC	0.0225 s, 33 it.	2.77 s, 72 it.
Physically Based - DC	0.0209 s, 33 it.	2.67 s, 77 it.
M-Block - DC	0.0634 s, 7 it.	2.50 s, 72 it.

of storage needed for the matrix. This method was tested as well with similar results, but the results are not shown here because the standard storage allows an easier comparison for the preconditioners. Such a storage scheme would show its greatest benefit on much large systems, which were not studied here.

The M-block and physically based diagonal transmissivity preconditioners perform the best for the homogenized source sink problem with about 4-12 times the speed up of the stand alone CG method. The direct Cholesky (DC) preconditioners perform better than the incomplete Cholesky. It should be noted, however, that the direct Cholesky also has a larger upfront cost which isn't included in the calculations here. Only the time to go

through the CG iterations, not any upfront cost is shown. So the direct Cholesky would be better for problems that get repeated multiple times for multiple right hand sides and the incomplete Cholesky would likely perform better for problems with fewer repetitions of the same domain. The Jacobi or diagonal preconditioner, similar to the results in Chapter 2, actually causes the system not to converge in some cases.

3.3.3 Single Phase Flow Problem

The preconditioners were also tested on a 1D flux problem where the right hand side is set to zero, but there is assumed to be a unit velocity at the boundaries in one of the coordinate directions. The resulting pressure is shown in Figure 3.7 and the results for the two dimensional and three dimensional problems are shown in Tables 3.3 and 3.4.

Pressure for Single Phase Flow Problem

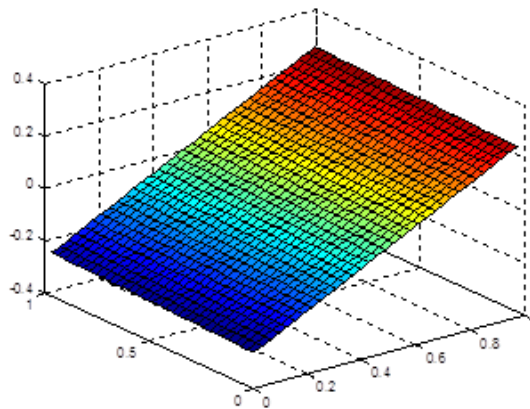


Figure 3.7. Resulting Pressure for MFEM Single Phase Flow Problem with $m = n = 33$

Table 3.3. Results of Preconditioning on MFEM 2D Single Phase Flow Problem

Method	Mesh Size (<i>m</i> -spatial, <i>n</i> -spatial)		
	<i>m</i> = 45, <i>n</i> = 45	<i>m</i> = 65, <i>n</i> = 65	<i>m</i> = 85, <i>n</i> = 85
SSOR	0.601 s, 64 it.	3.68 s, 92 it.	13.8 s, 119 it.
None - CG	0.433 s, 190 it.	2.64 s, 276 it.	9.82 s, 341 it.
Jacobi (Diagonal)	0.475 s, 201 it.	2.92 s, 294 it.	11.0 s, 387 it.
None - DC	0.231 s	1.40 s	6.50 s
Block Diag - IC	0.351 s, 140 it.	2.04 s, 193 it.	7.51 s, 252 it.
M-Block - IC	0.221 s, 81 it.	1.35 s, 116 it.	4.72 s, 150 it.
Physically Based - IC	0.221 s, 81 it.	1.30 s, 116 it.	4.77 s, 150 it.
M-Block - DC	0.229 s, 81 it.	1.29 s, 116 it.	4.78 s, 150 it.
Physically Based - DC	0.224 s, 81 it.	1.29 s, 116 it.	4.73 s, 150 it.

Here as well the physically based and M-block preconditioners perform the best with about 2-8 times the speed up of the stand alone CG method.

3.3.4 1D Flux Varied Transmissivity Problem

The final problem is the same as the previous problem except that the transmissivity tensor diagonal is set and the magnitude of the off-diagonal term is varied to see the effect on the preconditioner. For example, we have

$$T_H = \begin{bmatrix} 10 & \alpha \\ \alpha & 10 \end{bmatrix} \quad (3.63)$$

Table 3.4. Results of Preconditioning on MFEM 3D Single Phase Flow Problem

Method	Mesh Size	
	<i>(m-spatial in x, n-spatial in y, o-spatial in z)</i>	
	<i>m = 10, n = 10, o = 10</i>	<i>m=20,n=20,o=20</i>
Jacobi (Diagonal)	0.312 s, 597 it.	DNC
None - CG	0.125 s, 213 it.	11.6 s, 325 it.
None - DC	0.0331 s	12.2 s
Block Diag - IC	0.0378 s, 66 it.	3.52 s, 96 it.
SSOR	0.0485 s, 35 it.	2.45 s, 45 it.
Physically Based - IC	0.0158 s, 24 it.	1.51 s, 41 it.
M-Block Diag - IC	0.0181 s, 25 it.	1.54 s, 41 it.
Physically Based - DC	0.0158s, 24 it.	1.55 s, 41 it.
M-Block - DC	0.0166 s, 25 it.	1.59 s, 41 it.

for different values of α ranging from zero to ten. The results for the two dimensional problem is shown in Table 3.5. Here we are only testing the physically based incomplete Cholesky preconditioner against the stand alone CG method and showing the change in ratio of speed up in time and number of iterations.

Here we also see the improvement from using the preconditioner. As expected, for initial smaller values of alpha with magnitude only slightly greater than zero, the preconditioning performs better than for values closer to the diagonal value of ten.

Table 3.5. Results of Preconditioning on MFEM 2D Varied Transmissivity Test

Value of Alpha	Run Times and Iterations for $m = 65, n = 65$		
	CG	PB IC	Ratios
-1	2.94 s, 306 it.	1.57 s, 138 it.	1.87, 2.22
-2	3.08 s, 317 it.	1.62 s, 145 it.	1.90, 2.19
-3	3.24 s, 332 it.	1.71 s, 153 it.	1.89, 2.17
-4	3.31 s, 341 it.	1.76 s, 159 it.	1.88, 2.14
-5	3.35 s, 345 it.	1.93 s, 172 it.	1.74, 2.01
-6	3.60 s, 374 it.	2.13 s, 186 it.	1.69, 2.01
-7	3.77 s, 395 it.	2.35 s, 210 it.	1.60, 1.88
-8	4.37 s, 455 it.	2.61 s, 238 it.	1.67, 1.91
-9	5.60 s, 557 it.	3.37 s, 309 it.	1.66, 1.80
-9.9	9.22 s, 929 it.	6.27 s, 576 it.	1.47, 1.61
-9.99	10.7 s, 1084 it.	7.78 s, 695 it.	1.38, 1.56

This is due to the fact that for small values of α , A is still block diagonally dominant making M , which is sparse block diagonally dominant, a good approximation of A . As alpha increases, the off-diagonal blocks gain more significance and M becomes a weaker approximation of A . Even for values close to 10, however, the preconditioner still offers close to one and a half times the speed up.

3.4 Conclusions and Future Work

As can be seen in the results, improvements can be made on the run times for linear systems that result from the mixed finite element approximations of the flow equation in porous mediums. Three levels of improvement were presented including homogenization of the porous medium domain, a projection method for flow equations with full transmissivity tensors, and preconditioning of the linear system resulting from the projection method. A fourth improvement, parallelization, will be discussed in Chapter 4. The physically based preconditioner has been shown to be effective in reducing the run time and number of iterations for the preconditioned CG method. The physically based preconditioner also is comparable if not better than several standard preconditioners. The physically based preconditioner was shown to be affected by changes in the diagonal dominance of the transmissivity tensor, but still showed improvement on the CG method for less diagonally dominant matrices.

This project was not an exhaustive result of all preconditioners or linear solution methods and further results can be done in each area. Other linear solution methods that could be compared are generalized minimum residual methods and more especially multigrid methods, which, although complex, have been shown to be the fastest linear solvers [55]. As seen above there are also many ways of implementing the preconditioners which could be explored further. Further work could also be done on larger parallel computing architectures to see the results of each method there. This will be explored in some detail in chapter 4, but only as it relates to parallel computing on graphics cards using CUDA.

In summary, the physical based preconditioning proved effective on improving the CG linear solver of the mixed finite element formulations of the flow equation with full transmissivity tensors and further studies could be done to determine better linear system solution methods or preconditioners and preconditioner methods. In the next chapter we will see the results of a physically based preconditioner on a parallel architecture in CUDA.

CHAPTER 4

PRECONDITIONING FOR FINITE ELEMENT METHODS APPLIED TO FIRST ORDER PARTICLE TRANSPORT AND TO FLUID TRANSPORT IN POROUS MEDIA IMPLEMENTED IN PARALLEL ON GPUS

Four levels of improvement were used to increase the efficiency of solving the flow problem in porous media, Equation (3.1), using the mixed finite element method. The first and second have been done previously which are the homogenization of the transmissivity tensor and the projection method to modify the mixed finite element method. Two more levels of improvement are applied here, preconditioning of the linear system resulting from the mixed finite element method and preconditioning of the conjugate gradient solver in parallel in CUDA. The preconditioning of the linear system was shown in Chapter 3 and the algorithms for the linear solvers were shown in Chapter 2. In this chapter, we will be showing the final level of improvement, implementing the preconditioned conjugate gradient linear solver and the physically based preconditioner for the fluid transport problem in CUDA and comparing it with algebraic preconditioners. We will also show the results of using the physically based un-collided flux preconditioner for conjugate gradient linear solver of the least squares finite element method (LSFEM) of the particle transport problem in CUDA compared with algebraic preconditioners.

These conjugate gradient methods were run on a relatively new software language, CUDA (Compute Unified Device Architecture), developed by NVIDIA for processing on graphics processing units (GPU's). There are currently several languages for processing

on GPU's which also include OpenCL and OpenGL. The experiments for this project were run on an NVIDIA GEFORCE 610M graphics processing unit so CUDA was chosen since CUDA was designed specifically to run on NVIDIA graphics processing units.

CUDA, and scientific computations on GPU's, are relatively new. GPU's provide inexpensive, generally available, massively parallel computing hardware [56]. CUDA with other languages have made it easier to utilize this hardware. Any computer with a monitor has a GPU. It is just a matter of how many individual compute units are available. In addition, any number of GPUs can be added via USB connections to a computer. So programs in CUDA are widely accessible at little or no extra cost to execute them. When talking about GPUs two natural question arise. The first is whether or not GPU computing is competitive with or better than single core computers, multi-core computers or traditional parallel machines. In recent results, GPU computing has been shown to surpass multi-core computations on a number of applications [56]. The second question is whether certain algorithms that run well in serial also run well in parallel on GPUs.

Of particular interest to this research is how these questions are answered for GPU computing applied particle or fluid transport applications. A number of studies have been done to show the utility of GPU computing for particle transport problems. One very natural application was utilizing GPU computing for Monte Carlo methods applied to particle transport problems. Monte Carlo methods are derived from the probabilities associated with the cross sections of the materials through which particles may be traveling. In other words, one at a time, particles are tracked as they travel through or get

absorbed by a material. When a particle interacts with a material at a given point, a probability is used to determine the type of interaction (i.e. absorption, scattering, no interaction, etc.) and the angle of scattering if any. In order to develop an accurate assessment of the general transport of particles through a material, a large number of single particle simulations must be run. These single particle simulations can be done independently, making Monte Carlo simulations inherently parallel problems. In [13], [57], and [58] CUDA was used as part of Monte Carlo particle transport applications. They each show good speed-ups when working with GPU's.

Some work has also been done with deterministic solution methods like the finite element methods in this paper. Papers [14] and [59], and [60] show results of GPU computing on deterministic applications including a discrete ordinates method, a method of characteristics, and a source-iteration method. Studies have also been done that utilize both the cpu and the GPU. Such a hybrid method for a deterministic transport code is shown in [60]. All of these studies report good speedups when using GPU computing.

GPU computing has also been explored in the realm of fluid flow in porous media. In [15] a homogenization method for heterogeneous media was applied using CUDA. Work on multiple GPUs was done in [16] for a natural porous media problem and in [61] a hybrid CPU-GPU method was used for a two-phase porous media problem. Each of these also show good speed-ups when using GPU computing.

GPU computing is also used more generally to improve iterative methods for linear systems including the conjugate gradient method. In [62] an overview study is done on GPU computing for the preconditioned conjugate gradient method. Their conclusion was summarized: "Based on the experimental results...we observe that, when used as general

purpose many-core processors, current GPUs provide a much lower performance advantage for irregular (sparse) computations than they can for more regular (dense) computations...however, when used carefully, GPUs can still be beneficial as co-processors to CPUs to speed-up complex computations." In other words, GPU computing has its limits, and while it has proven to be generally faster than single core and some multi-core computers, it is still slower than traditional multi-core computing clusters for some problems.

In [17] and [18], preconditioned iterative methods are run in parallel on GPUs including algebraic preconditioners. In [17] an SSOR type preconditioner is used and in [18] a sparse approximate inverse preconditioners is used based on the singular values of the linear system matrix. Hybrid methods including cpu and GPU computations were applied in [63] and [64] where the conjugate gradient method was applied on multiple GPU platforms. In [65], a bi-conjugate gradient method for a finite difference approximation was also tested in CUDA on GPUs.

In this work we extend the use of GPU computing to the least squares finite element method (LSFEM) applied to the first order particle transport equation of Chapter 2 and the projected mixed finite element method (MFEM) applied to the homogenized fluid transport equation of Chapter 3, seeking to answer the second question of whether the physically based preconditioners are effective when run in parallel. Specifically we show results of running the conjugate gradient linear solver in CUDA on the GPU for each problem and the results of using the physically based preconditioners of each problem in CUDA on the GPU. For both problems, the physically based preconditioners perform well on the GPU giving speed-ups from about 2 to 50 times.

We will first give a brief background on processing with GPUs and some details on the implementation of the preconditioned conjugate gradient method in CUDA for the particle and fluid transport problems. We will then look at the results of running the preconditioned conjugate gradient method for the particle and fluid transport problems, specifically comparing the physically based preconditioner with algebraic preconditioners.

4.1 GPU Computing

GPU computing started out as a way to speedup computer graphics applications, largely for graphics in video games. It has grown to a wide variety of applications including medical imaging, computational fluid dynamics, environmental science [12] [66]. GPUs started out as 2D display accelerators offering hardware assisted bitmap operations. In 1992, OpenGL, a computing language for graphics cards, was introduced [12]. The video game industry continued to drive this new area and eventually NVIDIA and others added new capabilities to the GPU hardware as well as adding to the software with new software languages like CUDA and OpenCL. As seen above in [60] and [61], this area has extended to hybrid CPU-GPU hardware architectures and computing libraries to run on them.

4.2 Preconditioned Conjugate Gradient Method in CUDA

The preconditioned conjugate gradient methods for the least squares finite element method for first order particle transport and for the projected mixed finite element method for fluid transport in porous media were applied in CUDA using the CUDA

Toolkit, specifically the CUDA Basic Linear Algebra Subprograms (CUBLAS) library and the CUSPARSE library [12]. These CUDA libraries include several examples of linear algebra algorithms and iterative solvers. The toolkit and libraries for CUDA can be found on NVIDIA's website. For this work, the conjugate gradient method example within the CUDA SDK was modified to run with the sparse and full matrices of the particle transport and fluid transport problems.

The existing compressed sparse row (CSR) format within the example was used for the sparse matrices of the fluid and particle transport problems. The CSR format was modified to the full matrix format of CUBLAS for the projected mixed finite element method for the fluid transport problem and the functions for matrix multiplication in CUBLAS and CUSPARSE were changed accordingly. The existing incomplete Cholesky preconditioner format was used for each of the preconditioners. General input and output C++ libraries as well as some standard code was also added for determining the run times of the codes and checking the results.

For both the particle and fluid transport problems the matrices and vectors were generated from the Matlab codes mentioned in Chapters 2 and 3 and, together with their preconditioners, were transferred and run on the parallel form of the preconditioned conjugate gradient method example within the CUDA SDK. The tolerance was set to be the same as before, 10^{-9} and the initial guess set to be a vector of all zeros. More details on the implementation of the preconditioned conjugate gradient method can be found in the Appendix. The specific problems tested include the least squares finite element method (LSFEM) for first order particle transport in one and two dimensions and the

projected mixed finite element method for the homogenized fluid transport source sink problem. These are the problems that use the preconditioned conjugate gradient method.

4.3 Source Void LSFEM First Order Particle Transport Results

We will first look at the one dimensional Reed Problem 2.6.1 and then at the square source void problem 2.6.2 using the LSFEM. Recall that these particle transport problems are source void problems. The initial conditions and geometries can be found in Sections 2.5.1 and 2.5.2 respectively. The solvers include the stand alone conjugate gradient method (CG-None), the physically based incomplete Cholesky (PB IC) CG method, the algebraic M-Block (i.e. preconditioner derived from original matrix A with same structure as the physically based preconditioner M , see Section 2.4.3.2) CG method, and the incomplete Cholesky method from the original CUDA CG example. The results for each are shown in Figures 4.1 and 4.2 and in Tables 4.1 and 4.2 respectively.

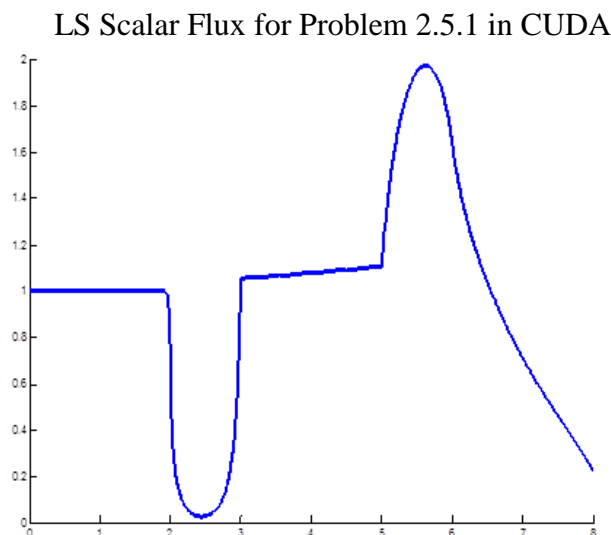


Figure 4.1. Resulting Flux for the Reed Problem Using the Continuous LSFEM in CUDA with $m = 16$, $n = 1600$

LS Scalar Flux for Problem 2.5.2 in CUDA

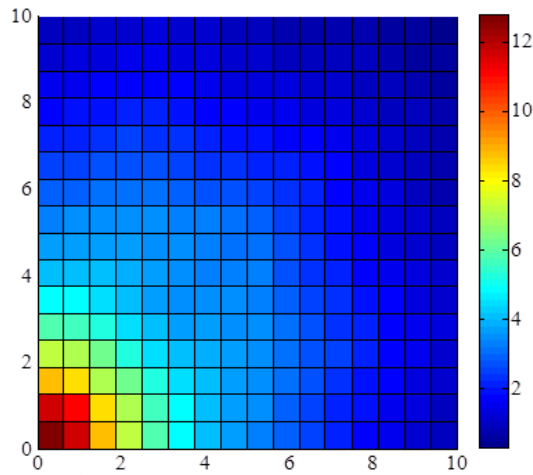


Figure 4.2. Resulting Flux for the Square Source Void Problem Using LSFEM in CUDA for S10

As can be seen in Tables 4.1 and 4.2, the physically based preconditioner for the one and two dimensional problems work well in parallel on the GPU as well. For the one dimensional problem the speedups range from fifty to over one hundred times. For the two dimensional problem speedups range from about four to five times. For the one dimensional problem, the physically based incomplete Cholesky (PB IC) preconditioner was compared with two algebraic preconditioners, the standard incomplete Cholesky (IC) and the M-Block incomplete Cholesky. The M-Block, as mentioned in Chapter 3, is a simplified version of the original system matrix A with the same structure as the preconditioner M . In this parallel CUDA case, the physically based preconditioner performs better than the standard Incomplete Cholesky and about equal to the M-Block preconditioner. For the two dimensional problem, the physically based preconditioner was only compared to the M-Block preconditioner due to memory constraints for the standard incomplete Cholesky preconditioner.

Table 4.1. Parallel Results of Preconditioning on the LSFEM for Reed Problem

Method	Mesh Size (n -spatial, m -direction)		
	$m = 16, n = 800$	$m = 16, n = 1600$	$m = 16, n = 2400$
Cuda - CG - None	5.54 s, 8133 it.	20.0 s, 17194 it.	41.4 s, 25777 it.
Cuda - IC	0.169 s, 8 it.	0.420 s, 10 it.	0.754 s, 12 it.
Cuda - M-Block IC	0.109 s, 23 it.	0.203 s, 23 it.	0.327 s, 25 it.
Cuda - PB IC	0.109 s, 24 it.	0.209 s, 24 it.	0.310 s, 24 it.

Table 4.2. Parallel Results of Preconditioning on the LSFEM for Square Source Void Problem

Method	Angular Quadratures for 16x16 Mesh		
	S8	S10	S12
Cuda - CG	1.58 s, 543 it.	3.92 s, 612 it.	7.94 s, 670 it.
Cuda - PB IC	0.438 s, 101 it.	0.874 s, 106 it.	1.53 s, 107 it.
Cuda - M-Block IC	0.462 s, 101 it.	0.858 s, 103 it.	1.51 s, 106 it.

In Tables 4.3 and 4.4, we compare the parallel version of the CG method and the physically based preconditioner with the original serial version. In general the serial version performs better on the one dimensional problem except for the two larger mesh sizes for the standalone CG method. This is not unexpected due to the simple nature of the one dimensional problem. For the two dimensional problem, the parallel version of the CG method and the physically based preconditioner generally perform better than the serial version, with about one and a half times speedup for the standalone CG method and

just slightly over one times speedup for the physically based incomplete Cholesky preconditioner.

Table 4.3. Comparison of Parallel and Serial Results for LSFEM on Reed Problem

Method	Mesh Size (n -spatial, m -direction)		
	$m = 16, n = 800$	$m = 16, n = 1600$	$m = 16, n = 2400$
Cuda - CG - None	5.54 s, 8133 it.	20.0 s, 17194 it.	41.4 s, 25777 it.
Cuda - PB IC	0.109 s, 24 it.	0.209 s, 24 it.	0.310 s, 24 it.
None - CG	4.811 s, 8106 it.	20.64 s, 17171 it.	45.67 s, 25763 it.
PB - IC	0.0227 s, 23 it.	0.0469 s, 22 it.	0.0790 s, 22 it.

Table 4.4. Comparison of Parallel and Serial Results for LSFEM on Square Source Problem

Method	Angular Quadratures for 16x16 Mesh		
	S8	S10	S12
Cuda - CG	1.58 s, 543 it.	3.92 s, 612 it.	7.94 s, 670 it.
Cuda - PB IC	0.438 s, 101 it.	0.874 s, 106 it.	1.53 s, 107 it.
CG	2.28 s, 511 it.	5.53 s, 586 it.	11.5 s, 648 it.
PCG - PB IC	0.491 s, 98 it.	1.06 s, 104 it.	2.02 s, 106 it.

4.4 Projected MFEM Source Sink Fluid Transport Results

Here we will first look at the results of the two dimensional source sink problem of Section 3.7.2 and then at the three dimensional source sink problem. Recall that this is the

projected mixed finite element method solution on the homogenized particle transport problem. The solvers include the stand alone conjugate gradient method (CG-None), the physically based incomplete Cholesky (PB IC) CG method, the algebraic M-Block CG method, and the block diagonal incomplete Cholesky method. The results for the two dimensional problem are shown in Figure 4.3 and Table 4.5 and the results for the three dimensional problem are shown in Table 4.6.

MFEM Pressure for 2D Source Sink Problem in CUDA

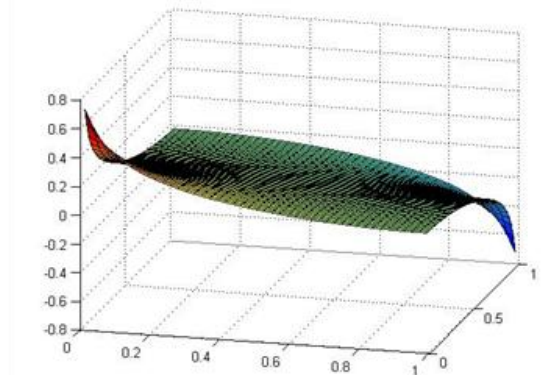


Figure 4.3. Resulting Pressure for MFEM Source Sink Problem in CUDA with $m = n = 45$

The physically based preconditioner performs well providing about two times speedup for the two dimensional problem and about five to eight times speedup for the three dimensional problem. As seen in Table 4.5 and 4.6, the physically based preconditioner was compared against two algebraic preconditioners, the M-Block incomplete Cholesky and the block diagonal incomplete Cholesky. The physically based preconditioner performs the best. It is about equal to the M-Block incomplete Cholesky for two of the mesh sizes, but converges when the M-Block does not for the largest mesh size. It is unclear exactly why the M-Block did not converge for the largest mesh size, but

as mentioned in Chapter 2, iterative solvers like the conjugate gradient method can stagnate depending on the mesh size and machine precision. For the three dimensional problem, the physically based preconditioner performs about equal to the M-Block preconditioner, with the M-Block perhaps slightly better.

Table 4.5. Parallel Results of Preconditioning on the MFEM 2D Source Sink Problem

Method	Mesh Size (m -spatial in x , n -spatial in y)		
	$m = 45, n = 45$	$m = 65, n = 65$	$m = 85, n = 85$
CUDA - CG - None	0.564 s, 238 it.	3.33 s, 346 it.	12.8 s, 452 it.
CUDA - BD IC	0.679 s, 174 it.	2.94 s, 253 it.	10.3 s, 332 it.
CUDA - M-Block IC	0.344 s, 83 it.	1.50 s, 128 it.	DNC
CUDA - PB IC	0.343 s, 83 it.	1.56 s, 128 it.	5.16 s, 157 it.

Table 4.6. Parallel Results of Preconditioning on the MFEM 3D Source Sink Problem

Method	Mesh Size (m -spatial in x , n -spatial in y)	
	$m = 10, n = 10, o = 10$	$m = 20, n = 20, o = 20$
CUDA - CG - None	0.203 s, 309 it.	15.8 s, 672 it.
CUDA - BD IC	0.140 s, 99 it.	4.03 s, 160 it.
CUDA - PB IC	0.047 s, 33 it.	1.99 s, 77 it.
CUDA - M-Block IC	0.047 s, 33 it.	1.87 s, 72 it.

In Tables 4.7 and 4.8, the parallel version of the CG method and the physically based preconditioner are compared with the serial versions. For the two dimensional problem, the parallel versions generally perform better than the serial versions with up to about

twice the speedup. For the three dimensional problem, the serial version performs slightly better than the parallel version for the smaller mesh size and the parallel version performs slightly better for the larger mesh size. This trend would be expected to continue for larger mesh sizes that couldn't be tested due to memory constraints.

Table 4.7. Comparison of Parallel and Serial for MFEM 2D Source Sink Problem

Method	Mesh Size (m -spatial in x , n -spatial in y)		
	$m = 45, n = 45$	$m = 65, n = 65$	$m = 85, n = 85$
CUDA - CG - None	0.564 s, 238 it.	3.33 s, 346 it.	12.8 s, 452 it.
CUDA - PB IC	0.343 s, 83 it.	1.56 s, 128 it.	5.16 s, 157 it.
None - CG	0.897 s, 396 it.	5.36 s, 575 it.	21.2 s, 752 it.
Physically Based - IC	0.266 s, 100 it.	1.64 s, 145 it.	5.89 s, 191 it.

Table 4.8. Comparison of Parallel and Serial for MFEM 3D Source Sink Problem

Method	Mesh Size (m -spatial in x , n -spatial in y)	
	$m = 10, n = 10, o = 10$	$m = 20, n = 20, o = 20$
CUDA - CG - None	0.203 s, 309 it.	15.8 s, 672 it.
CUDA - PB IC	0.047 s, 33 it.	1.99 s, 77 it.
None - CG	0.132 s, 275 it.	18.1 s, 510 it.
Physically Based - IC	0.0210 s, 33 it.	2.86 s, 77 it.

4.5 Conclusions and Future Work

In Chapter 3 we discussed four levels of improvement for mixed finite element method applied to the porous media fluid transport problem: a homogenization method, a projection of the mixed finite element method, a physically based preconditioner, and a parallel implementation of the linear solver on GPUs using CUDA. The first three improvements were shown in detail in Chapter 3. Here we showed the fourth level of improvement, the implementation of the linear solver on GPUs using CUDA. Specifically, we modified the preconditioned conjugate gradient method example within the CUDA SDK to run the CG method and preconditioned CG with the inputs from the projected mixed finite element method for the homogenized porous media fluid transport problem.

The physically based preconditioner was shown to be effective in the parallel CUDA GPU linear solver providing about five to eight times the speedup to the standalone CG solver. The parallel code also generally performed better than the serial version of Chapter 3 with slightly better run times. The parallel preconditioned conjugate method was also used on the least squares finite element method applied to the first order particle transport problem. The physically based un-collided flux preconditioner also showed good speedup compared with the standalone CG method, and the parallel code showed some improvement on the serial version of Chapter 2, especially on the two dimensional code.

Further study could be done to fully optimize the parallel CUDA linear solver as well as a study of the biconjugate gradient method for the non-symmetric discontinuous finite element method. The main focus here was on the effectiveness of the

preconditioner in parallel, but other codes were written in Matlab and C++ that save memory by not saving the full linear system matrix, but instead only storing the operation of the matrix. These operations were tested as part of the study of Chapter 3, but results were not included. A parallel version of the matrix operation could also be written and tested within the CUDA framework. More complex domains for the original problems could also be studied as well as the optimal platform for the first order particle and porous media fluid transport problems utilizing multi-gpu and hybrid cpu-gpu.

In summary, the physically based preconditioners of the fluid and particle transport problems were shown to be effective in parallel computations on GPUs using CUDA, and the parallel CUDA codes were shown to be slightly better than the serial codes on the CPU. Further studies could be done as to the optimal platform (multi-gpu, hybrid gpu-cpu, multicore cpu, etc.) and memory storage for running the preconditioned conjugate gradient method in parallel.

CHAPTER 5

CONCLUSION

Physically based preconditioning was used to improve the efficiency of the linear solvers for two applications, first order particle transport and fluid transport in porous media. This preconditioning was also tested in parallel on GPUs using CUDA. In all cases the physically based preconditioner performed well, in terms of speed-up gained and as compared with several algebraic preconditioners. We also reviewed first order formulations of the neutron transport equation, an alternative to second order formulations, and two finite element implementations for the first order formulation to which the physically based un-collided flux preconditioner was applied. To the mixed finite element method for the simple flow equation for porous media flows, four levels of improvement were applied: the method of homogenization, a projection method, physically based preconditioning, and parallel implementation on GPUs. In summary, we extended the results of the LSFEM of [7] and the DFEM of [8] to include physically based preconditioning and implementation on GPUs. We also extended the results of [2] and [11] to include physically based preconditioning and parallel implementation on GPUs. Future work could include applying the mixed finite element method to the first order particle transport equation, testing other linear solvers on these problems, especially multigrid solvers, implementing these methods on more complex hybrid cpu-gpu architectures like those in [16] [61] [63] and [64], or in more fully optimizing the code through different storage schemes and algorithms that do not require storage of the linear system matrix.

REFERENCES

1. R. Strzodka, J. Cohen, and S. Posey. "GPU-Accelerated Algebraic Multigrid for Applied CFD," *Procedia Engineering*, **61**, pp. 381-387 (2013).
2. M. T. Heath. *Scientific Computing: An Introductory Survey, Second Edition*, McGraw-Hill Companies, Inc. (2002).
3. J. L. Liscum-Powell, W. J. Bohnoff, C. R. Drumm, and W. C. Fan. "CEPTRE/Nevada Physics Guide Version 1.0," Sandia Report SAND2007-7409, Sandia National Laboratories (2007).
4. J. E. Morel et al. "Spatial discretizations for self-adjoint forms of the radiative transfer equations," *Journal of Computational Physics*, **214** (1), pp. 12-40 (2006).
5. L. Cao and H. Wu. "A spherical harmonics--Finite element discretization of the self-adjoint angular flux neutron transport equation," *Nuclear Engineering and Design*, **237** (23), pp. 2232-2239 (2007).
6. M. L. Adams and E. W. Larsen. "Fast Iterative Methods for Discrete-Ordinates Particle Transport Calculations," *Progress in Nuclear Energy*, **40** (1), pp.3-159 (2002).
7. C. Drumm and W. Fan, "Least Squares Finite Element Algorithms in the SCEPTRE Radiation Transport Code," *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011)*, Rio de Janeiro, RJ, Brazil, May 8-12, on CD-ROM, Latin American Section (LAS) / American Nuclear Society (ANS) ISBN 978-85-6368-00-2 (2011).

8. T. Manteuffel, S. McCormick, J. Morel, S. Oliveira, and G. Yang, "A Fast Multigrid Algorithm for Isotropic Transport Problems I: Pure Scattering," *SIAM J. Sci. Comput.*, **16(3)**, pp.601-635 (1995).
9. R. E. Ewing and M. F. Wheeler, *Computational Aspects of Mixed Finite Element Methods*, North-Holland, Amsterdam (1983).
10. G. Amaziane, A. Bourgeat, and J. Koebbe. "Numerical Simulation and Homogenization of Two-Phase Flow in Heterogeneous Porous Media," *Transport in Porous Media*, **6**, pp. 519-547 (1991).
11. J. Koebbe, "A Computationally Efficient Modification of Mixed Finite Element Methods for Flow Problems with Full Transmissivity Tensors," *Numerical Methods for Partial Differential Equations*, **9**, pp.339-355 (1993).
12. *CUDA C PROGRAMMING GUIDE*, NVIDIA Corporation, 2007-2013.
13. X. Jia, et al. "Development of a GPU-based Monte Carlo dose calculation code for coupled electron-photon transport," *Phys. Med. Biol.*, **55 (11)**, pp. 3077-3086 (2010).
14. C. Gong, J. Liu, L. Chi, H. Huang, J. Fang, and Z. Gong. "GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method," *Journal of Computational Physics*, **230 (15)**, pp. 6010-6022 (2011).
15. B. Quintela, D. Caldas, M. Farange, and M. Lobosco. "Multiscale Modeling of Heterogeneous Media Applying AEH to 3D Bodies," *Computational Science and its Applications - ICCSA 2012, Lecture Notes in Computer Science*, **7333**, pp. 675-690 (2012).
16. S. Ovaysi and M. Piri. "Multi-GPU acceleration of direct pore-scale modeling of fluid flow in natural porous media," *Computer Physics Communications*, **183 (9)**, pp. 1890-1898 (2012).

17. R. Helfenstein, J. Koko. "Parallel preconditioned conjugate gradient algorithm on GPU," *Journal of Computational and Applied Mathematics*, **236 (15)**, pp. 3584-3590 (2012).
18. M. Grote, and T. Huckle. "Parallel Preconditioning with Sparse Approximate Inverses," *SIAM J. Sci. Comput.*, **18 (3)**, 838-853 (1997).
19. M. Rigley and C. Drumm. "Matrix Preconditioning for Photon Transport Equations," Technical Report, Sandia National Laboratories, SAND2011-6529 P (2011).
20. M. A. Heroux and J. M. Willenbring. "Trilinos Users Guide," Technical Report, SAND2003-2952, Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550 (2003).
21. M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, and M. G. Sala. "ML 5.0 smoothed aggregation user's guide, Technical Report SAND2006-2649, Sandia National Laboratories (2006).
22. M. Rigley and C. Drumm. "Matrix Preconditioning For Neutron Transport Equations," Technical Report, Sandia National Laboratories, SAND2012-7676 P (2012).
23. R. T. Ackroyd and N. S. Riyait, "Iteration and Extrapolation Methods for the Approximate Solution of the Even-Parity Transport Equation for Systems With Voids," *Ann. nucl. Energy*, **16 (1)**, pp.1-32 (1989).
24. E. E. Lewis and W. F. Miller, Jr. *Computational Methods of Neutron Transport*, American Nuclear Society, La Grange Park, Illinois (1993).
25. T. A. Brunner. "Forms of Approximate Radiation Transport," Sandia Report, SAND2002-1778, Sandia National Laboratories (2002).
26. W. L. Morgan. "ELENDF: A time-dependent Boltzmann solver for partially ionized plasmas," *Computer Physics Communications*, **58 (1-2)**, pp. 127-152 (1990).

27. A. J. H. McGaughey and M. Kaviani. "Quantitative validation of the Boltzmann transport equation phonon thermal conductivity model under the single-mode relaxation time approximation," *Physical Review B*, **69**, pp. 094303 (2004).
28. M. S. Gockenbach. *Understanding and Implementing the Finite Element Method*, Society for Industrial and Applied Mathematicians (2006).
29. W. Reed, "New Difference Schemes for the Neutron Transport Equation," *Nucl. Sci. Eng.*, **46** (2), pp.309-314 (1971).
30. C. J. Gesh and M. L. Adams, "Even- and Odd-Parity Finite Element Solutions to Thick Diffusive Problems in Cartesian Geometry," *Advanced Methods in Radiation Transport, M&C 99*, Madrid, Spain (1999).
31. T. M. Austin and T. A. Manteuffel. "A Least-Squares Finite Element Method for the Linear Boltzmann Equation with Anisotropic Scattering," *SIAM J. Numer. Anal.*, **44** (2), pp. 540-560, Society for Industrial and Applied Mathematicians (2006).
32. M.E. Cantekin and J.J. Westerink, "Non-diffusive $N + 2$ Degree Petrov-Galerkin Methods for Two-Dimensional Transient Transport Computations," *International Journal for Numerical Methods in Engineering*, **30**, pp.397-418 (1990).
33. H. Anton and C. Rorres. *Elementary Linear Algebra: Applications Version, Eight Edition*, John Wiley & Sons, Inc. (1973).
34. Saad, Yousef. *Iterative Methods for Sparse Linear Systems*. Yousef Saad (2000).
35. G. H. Golub and C. F. Van Loan. *Matrix Computations, Third Edition*, The John Hopkins University Press (1996).
36. Picture Reference, Figure 2.4, http://hep.physics.indiana.edu/~hgevans/p410-p609/material/06_fit/func_min.html.

37. R. Moore. *Vector and Matrix Differentiation*, Ed. Ross Moore and Nikos Drakos, Macquarie University, Sydney, 1 Feb. 2002. Web. 22 Aug. (2011).
<http://fourier.eng.hmc.edu/e161/lectures/algebra/node7.html>
38. T. K. Moon and W. C. Sterling. *Mathematical Methods and Algorithms for Signal Processing*, Prentice Hall (2000).
39. Picture Reference, Figure 2.5. <http://www.dreamstime.com/stock-photography-mountain-trail-switzerland-alps-image6166482>.
40. H. A. Van Der Vorst. "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, **12**, pp. 631-644 (1992).
41. C. Drumm and W. Fan. "Uncollided-Flux Preconditioning of the Conjugate Gradients Solution of the Transport Equation," *Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew*, Gatlinburg, Tennessee, April 6-11, 2003, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2003).
42. G. L. G. Sleijpen and D. R. Fokkema. "BiCGstab(/) for linear equations involving unsymmetric matrices with complex spectrum," *Electronic Transactions on Numerical Analysis*, **1**, pp. 11-32 (1993).
43. Y. Watanabe and C.W. Maynard. "The discrete cones method in two dimensional neutron transport computations," University of Wisconsin, Report UWFD-574 (1984).
44. M. Rigley, J. Koebe, and C. Drumm, "Uncollided-flux Preconditioning for the First Order Transport Equation," *International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)*, Sun Valley, Idaho, USA, May 5-9, 2013, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2013).

45. P.A. Raviart and J.M. Thomas. "Mixed finite element methods for second-order elliptic problems," *Lecture Notes in Mathematics*, **606**, Springer-Verlag, New York (1977).
46. F. Brezzi, J. Douglas, Jr., and L.D. Marini. "Two Families of Mixed Finite Elements for Second Order Elliptic Problems," *Numerische Mathematik*, **47**, pp. 17-235, Springer-Verlag (1985).
47. T.F. Russell and M.F. Wheeler. "Finite Element and Finite Difference Methods for Continuous Flows in Porous Media," *The Mathematics of Reservoir Simulation*, the Society for Industrial and Applied Mathematicians (1983).
48. P. Colella, P. Concus, and J. Sethian. "Some Numerical Methods for Discontinuous Flows in Porous Media," *The Mathematics of Reservoir Simulation*, the Society for Industrial and Applied Mathematicians (1983).
49. M. H. Holmes. *Introduction to Perturbation Methods*, Beijing : Springer-Verlag, (1999). ISBN 7-5062-2682-0.
50. L. L. Watkins. *Using Wavelets as a Computational and Theoretical Tool for Homogenization*, Logan, UT : Utah State University (2005).
51. U. Hornung. "Homogenization and Porous Media," *Interdisciplinary Applied Mathematics* **6** (1997).
52. M. Rigley. "Homogenization for Porous Media," A technical report written in partial fulfillment of PhD Degree, Utah State University (2012).
53. C. Chen and E. Meiburg. "Miscible porous media displacements in the quarter five-spot configuration. Part 1. The homogeneous case," *J. Fluid Mech*, **371**, pp. 233-268, Cambridge University Press (1998).

54. C.W. Brand, U. Stanford, J.E. Heinemann, U. L. Mining, and K. Aziz. "The Grid Orientation Effect in Reservoir Simulation," SPE Symposium on Reservoir Simulation, 17-20 February, Anaheim, California (1991).
55. G. W. Milton. *The Theory of Composites*, Cambridge, UK: Cambridge University Press (2002).
56. R. Farber. *CUDA Application Design and Development*, Morgan Kaufmann (2011). ISBN-13: 978-0-12-388426-8.
57. A. Badal and A. Badano. "Accelerating Monte Carlo simulations of photon transport in voxelized geometry using a massively parallel graphics processing unit," *Med. Phys.*, **36**, pp. 4878 (2009).
58. F. A. van Heerden. "A Coarse Grained Particle Transport Solver Designed Specifically for Graphics Processing Units," *Transport Theory and Statistical Physics*, **41 (1)** (2012).
59. Z. Zhang and Q. Kan Wang. "Accelerating a three-dimensional MOC calculation using GPU with CUDA and two-level GCMFD method," *Annals of Nuclear Energy*, Elsevier, **62**, pp. 445-451 (2013).
60. C. Gong, J. Liu, H. Chen, J. Xie, and Z. Gong. "Accelerating the Sweep3D for a Graphic Processor Unit," *Journal of Information Processing Systems*, **7 (1)**, pp. 63-74 (2011).
61. M. Trapeznikova, B. Chetverushkin, N. Churbanova and D. Morozov. "Two-Phase Porous Media Flow Simulation on Hybrid Cluster," *Large-Scale Scientific Computing, Lecture Notes in Computer Science*, **7116**, pp. 646-653 (2012).
62. L. Ruipeng and Y. Saad. "GPU-Accelerated Preconditioned Iterative Linear Solvers," Technical Report, Department of Computer Science & Engineering; University of Minnesota, USA (2010).

63. M. Ament, G. Knittel, D. Weiskopf, and W. Strasser. "A Parallel Preconditioned Conjugate Gradient Solver for the Poisson Problem on a Multi-GPU Platform," *Parallel, Distributed and Network-Based Processing (PDP)*, 2010 18th Euromicro International Conference, 17-19 Feb., pp. 583-592 (2010).
64. A. Cevahir, A. Nukada, and S. Matsuoka. "Fast Conjugate Gradients with Multiple GPUs," *Computational Science - ICCS 2009, Lecture Notes in Computer Science*, **5544**, pp. 893-903 (2009).
65. G. Grawanis, C. Filelis-Papadopoulos, K. Giannoutakis. "Solving finite difference linear systems on GPUs: CUDA based Parallel Explicit Preconditioned Biconjugate Conjugate Gradient type Methods," *The Journal of Supercomputing*, **61 (3)**, pp 590-604 (2012).
66. J. Sanders, E. Kandrot. *CUDA BY EXAMPLE: An Introduction to General-Purpose GPU Programming*, NVIDIA Corporation (2011).

APPENDICES

APPENDIX A - CUDA TUTORIAL

Tutorial for Running Codes in CUDA

A.1 Installing CUDA in Windows

Directions for installing CUDA can be found on NVIDIA's website at

<http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/>

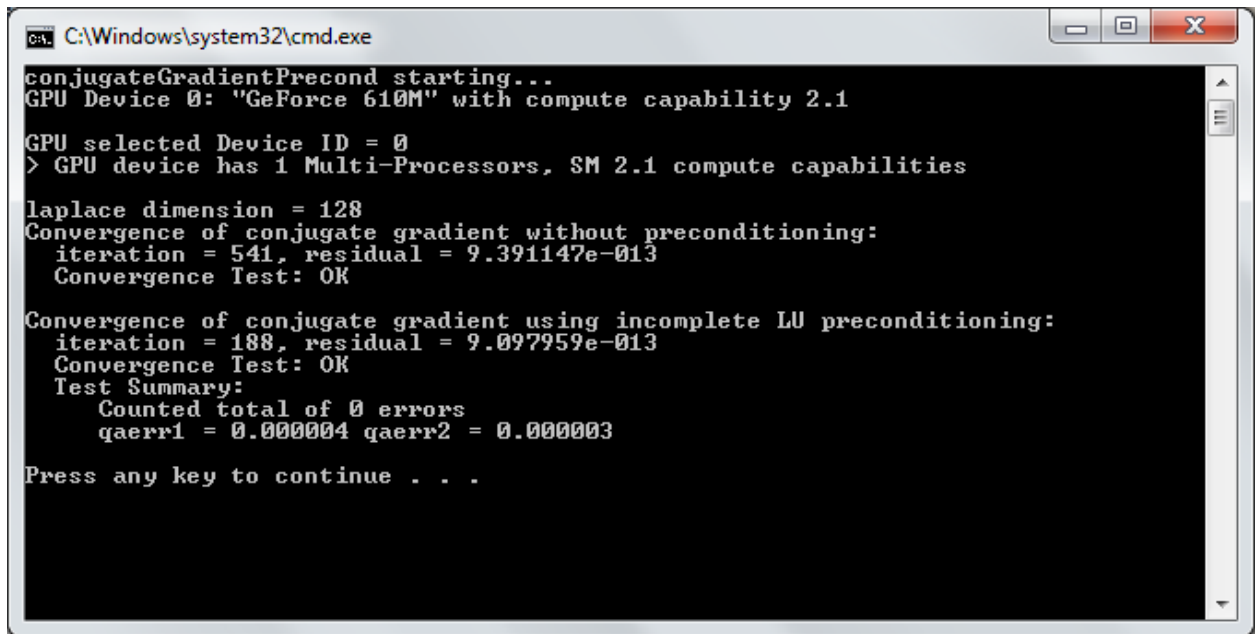
Additional helpful directions for using CUDA within a Visual C++ framework can be found at

<http://julip.co/2009/09/how-to-install-and-configure-cuda-on-windows/>

Once CUDA and Visual C++ are installed on your computer, you can find the preconditioned conjugate gradient method under the following folder

```
C:\ProgramData\NVIDIA\ Corporation\CUDA\Samples\...  
v5.0\7_CUDALibraries\conjugateGradientPrecond
```

Open this solution or project in Visual C++. The file that comes up should be main.cpp. This file contains code for running a conjugate gradient and preconditioned conjugate gradient method for a matrix made up of the Laplacian Operator with and incomplete LU factorization as the preconditioner. Run the code by typing ctrl + F5 to see the output. It should look like the figure below. Much of the code contained in the main.cpp file is for setting up the linear system to be run on the GPU and need not be changed to run the two iterative methods on a different linear system.



```

C:\Windows\system32\cmd.exe
conjugateGradientPrecond starting...
GPU Device 0: "GeForce 610M" with compute capability 2.1
GPU selected Device ID = 0
> GPU device has 1 Multi-Processors, SM 2.1 compute capabilities
laplace dimension = 128
Convergence of conjugate gradient without preconditioning:
iteration = 541, residual = 9.391147e-013
Convergence Test: OK
Convergence of conjugate gradient using incomplete LU preconditioning:
iteration = 188, residual = 9.097959e-013
Convergence Test: OK
Test Summary:
Counted total of 0 errors
qaerr1 = 0.000004 qaerr2 = 0.000003
Press any key to continue . . .

```

Figure A.1 Screen Shot of Results of CUDA Conjugate Gradient Method

A.2 Running the LSFEM on CUDA

Several things need to be changed from the original file to accommodate the LSFEM linear system. First, add additional header files for inputting and outputting matrices and vectors to and from text files and for checking the run time. More specifically, add the following to the list of include statements at the top of the file main.cpp.

```

// includes, additions
#include <iostream>
#include <fstream>
#include <time.h>
using namespace std;

```

The function void genLaplace will not be used for the new linear system and can be deleted if desired. The second and main change to the code is replacing the existing linear system with a linear system of your own. The code is setup for compressed sparse row format (CSR) and the matrix that you input should be in the same format. If so, then you can use the current variables given in the code and leave the code for transferring

memory to the GPU alone. Otherwise, additional changes will have to be made. More specifically, lines 183-198 in main.cpp can be replaced by your linear system. Modify the size of the system, number of nonzeros, row pointers, column indices, nonzero matrix values, initial guess values, and right hand side values according to your linear system. Below is the code that can be used to replace the lines above to create the LSFEM linear system.

```

/* Create My Own Matrix */
M = N = 640;
nz = 6908;
I = (int *)malloc(sizeof(int)*(N+1)); // csr row pointers for
                                     matrix A

float *Itest;
Itest = (float *)malloc(sizeof(float)*(N+1));
J = (int *)malloc(sizeof(int)*nz); // csr column indices for
                                     matrix A

float *Jtest;
Jtest = (float *)malloc(sizeof(float)*nz);
val = (float *)malloc(sizeof(float)*nz); // csr values for
                                     matrix A

x = (float *)malloc(sizeof(float)*N);
rhs = (float *)malloc(sizeof(float)*N);

for (int i = 0; i < N; i++)
{
    rhs[i] = 0.0; // Initialize RHS
    x[i] = 0.0; // Initial approximation of solution
}

// Test Additions
ifstream ffin1;
ffin1.open("Aval.txt");
for(int ii=0;ii<nz;ii++)
{
    ffin1 >> val[ii];
}
ffin1.close();
ifstream ffin2;
ffin2.open("Acol.txt");
for(int ii=0;ii<nz;ii++)
{
    ffin2 >> Jtest[ii];
    J[ii] = int(Jtest[ii])-1;
}
ffin2.close();
ifstream ffin3;
ffin3.open("Arptr.txt");
for(int ii=0;ii<N+1;ii++)
{

```

```

        ffin3 >> Itest[ii];
        I[ii] = int(Itest[ii])-1;
    }
    ffin3.close();
    ifstream ffin4;
    ffin4.open("rhs.txt");
    for(int ii=0;ii<N;ii++)
    {
        ffin4 >> rhs[ii];
    }
    ffin4.close();

```

Note that the size of the linear system M , and the number of nonzeros are input manually into the code. For this case, the maximum number of iterations should be at least 3000. Also note that the values for the matrix values, row pointers, column indices, and right hand side values need to be saved to files `Aval.txt`, `Arptr.txt`, `Acol.txt`, and `rhs.txt` respectively and stored in the given folder. The size of matrix and number of nonzeros above are for the case when the number of scattering directions is 8 and the number of steps is 80. Once the above modifications are made, the conjugate gradient algorithm should run and give the following output.

```

C:\Windows\system32\cmd.exe
conjugateGradientPrecond starting...
GPU Device 0: "GeForce 610M" with compute capability 2.1
GPU selected Device ID = 0
> GPU device has 1 Multi-Processors, SM 2.1 compute capabilities
Convergence of conjugate gradient without preconditioning:
iteration = 2202, residual = 9.932782e-013
Convergence Test: OK
Convergence of conjugate gradient using incomplete LU preconditioning:
iteration = 6, residual = 6.692627e-013
Convergence Test: OK
Test Summary:
Counted total of 0 errors
qaerr1 = 0.000580 qaerr2 = 0.000031
Press any key to continue . . . _

```

Figure A.2 Screen Shot of Results of LSFEM in CUDA

This completes the tutorial. From here, simple modifications explained above can be made for different linear systems. To change the preconditioner like in the results of Chapter 4, modifications need to be made to the transferring of data to the graphics processing unit. Some of those changes are not too difficult, but will not be discussed within this tutorial.

APPENDIX B - USERS MANUALS

B.1 Users Manuals for Particle Transport Codes

B.1.1 Function Explanations for the 1D Continuous LS Finite Element Toolbox

List of Scripts

reedproblemsetup – This is a script that sets up the parameters for the problem given in [3].

NT1DSimulation – This script runs the solver for the 1D equation.

createfileforc - This script runs a few lines that convert the matlab matrices and rhs to text files to be run in the proper CUDA folder

List of Functions

cootwostand – This is an extra function included if you want to convert a matrix from COO to standard format.

LegGaussquad – This function runs Gauss Legendre quadrature and gives the directions and weights for the scattering integral. For this version the weights add up to 2.

matvec_csr – This function performs matrix vector multiplication for a matrix in CSR format and a vector in standard format.

NT_1D_FEMIs – This function takes the input parameters from the setup scripts and creates the linear system matrix.

NT_1D_FEMIs_precM – This function creates the preconditioned matrix. It gives the same result as NT_1D_FEM_coo with sigma_s set to zero, but written simpler.

`preccg_csr` – This function runs a preconditioned conjugate gradient algorithm in CSR format. See [7] for details.

For an example on how to run the code, look at script `NT1DSimulation`.

B.1.2 Function Explanations for the 1D Discontinuous Finite Element Toolbox

List of Scripts

reedproblemsetup – This is a script that sets up the parameters for the problem given in [3].

clifproblemsetup – This is a simpler problem set up that was used to test the code.

NT1DSimulation – This script runs the solver for the 1D equation.

createfileforc - This script runs a few lines that convert the matlab matrices and rhs to text files to be run in the proper CUDA folder

List of Functions

biconjgradstab – This function runs the biconjugate gradient stabilized iterative solver. See Saad's book on iterative solvers.

blockLU_precM – This function finds the LU decomposition of the preconditioning matrix.

cootwostand – This is an extra function included if you want to convert a matrix from COO to standard format.

LegGaussquad – This function runs Gauss Legendre quadrature and gives the directions and weights for the scattering integral. For this version the weights add up to 2.

LUsolve_precM – This function solves the linear system for the preconditioner given the preconditioner in LU form.

matvec_csr – This function performs matrix vector multiplication for a matrix in CSR format and a vector in standard format.

NT_1D_FEM_coo – This function takes the input parameters from the setup scripts and creates the linear system matrix.

NT_1D_FEM_coo_precM – This function creates the preconditioned matrix. It gives the same result as NT_1D_FEM_coo with σ_s set to zero, but written simpler.

precbiconj – This function is a preconditioned stabilized biconjugate gradient iterative solver modified from the one found on Wikipedia.

For an example on how to run the code, look at script NT1DSimulation.

B.1.3 Function Explanations for the 2D Continuous LS Finite Element Toolbox

List of Scripts

bc_riyait - Applies the boundary conditions for the square source void problem

bc_riyait_norhs - A simplified form of bc_riyait

createfileforc - This script runs a few lines that convert the matlab matrices and rhs to text files to be run in the proper CUDA folder

NT2DSimulation_5_5 - The main script to run the code including setting up the matrix, solving the linear system, and graphing the results

List of Functions

matsparstocsr - A function that takes a Matlab sparse matrix A and outputs the vectors representing the nonzero values, row pointers, and column indices in CSR format

NT_2D_LS - A function that takes in the problem parameters and computes components of the linear system. The linear system is assembled inside of

NT2DSimulation_5_5 to help achieve the largest possible matrix given the current Matlab memory limits

NT_2D_LS - A modification of NT_2D_LS to break up the work of NT_2D_LS and assist in avoiding the Matlab memory limit

List of Pre-computed Objects

s8quad - quadrature weight as directions for the S^8 level-symmetric set

s10quad - quadrature weight as directions for the S^{10} level-symmetric set

s12quad - quadrature weight as directions for the S^{12} level-symmetric set

LS_nx16_ny16_s8 - Linear System elements for the 16 x 16 case with S^8 level-symmetric set

LS_nx16_ny16_s10 - Linear System elements for the 16 x 16 case with S^{10} level-symmetric set

LS_nx16_ny16_s12 - Linear System elements for the 16 x 16 case with S^{12} level-symmetric set

B.1.4 Function Explanations for the 2D Discontinuous Finite Element Toolbox

List of Scripts

NT2DSimulation_5_5 - The main script to run the code including setting up the matrix, solving the linear system, and graphing the results

The following are scripts that are repeated throughout NT_2D_DFEM for the various parts of the linear system matrix. They are divided by the terms in Equation (27) above.

first_order_mux_neg

first_order_mux_neg_rbc

first_order_mux_pos

first_order_muy_neg

first_order_muy_neg_rbc

first_order_muy_pos

rhs_script

zero_order_nonscat

zero_order_scatter

List of Functions

NT_2D_DFEM - Function that takes the problem parameters and creates the linear system

NT_2D_DFEM_# - Simplified form of NT_2D_DFEM to split up the work and speedup the process

List of Data Objects

s8quad - quadrature weight as directions for the S^8 level-symmetric set

s10quad - quadrature weight as directions for the S^{10} level-symmetric set

s12quad - quadrature weight as directions for the S^{12} level-symmetric set

DFEM_nx16_ny16_s8 - Linear System elements for the 16 x 16 case with S^8 level-symmetric set

DFEM_nx16_ny16_s10 - Linear System elements for the 16 x 16 case with S^{10} level-symmetric set

DFEM_nx16_ny16_s12 - Linear System elements for the 16 x 16 case with S^{12} level-symmetric set

B.2 Users Manuals for Fluid Transport Codes

B.2.1 Function Explanations for the 2D and 3D MFEM Toolboxes

List of Scripts

MFEMSimulation - Main script for setting up the problem, building the linear system, solving it and graphing the result

homogExample - same as MFEMSimulation except that it's specified for problem 3.2.1 above

homogalphatest - same as homogExample except that it's modified to run the alpha test in 3.2.3

onedflowExample - same as MFEMSimulation except that it's specified to run the one dimensional flow problem 3.2.2

List of Functions

MFEM_Full - Creates the components of the linear system (80) above

MFEM_Full_flow - same as MFEM_Full but with modified boundary condition for the one dimensional flow problem 3.2.2

precCG - preconditioned conjugate gradient algorithm

solveCG - conjugate gradient algorithm

VITA

Michael Rigley

PhD Student, Department of Mathematics and Statistics, Utah State University

Email: michael.rigley@aggiemail.usu.edu • Phone: (801) 388-9909

Academic Preparation

Brigham Young University	Mathematics Russian	B.S.—3.77 GPA, 2007
Utah State University	Applied Mathematics, Interdisciplinary	M.S.—3.92 GPA, 2009 PhD—3.94 GPA, 2013

Professional Experience

Technical Intern, Sandia National Laboratories-SEERI	2 years
Research Assistant, Utah State University – Modeling Fluid Flow in Mountain Lakes	1 year
Research Fellow, National Physical Science Consortium	2 years
Graduate Instructor, Department of Mathematics and Statistics	5 years

Current Research

- *Finite Element Method Solvers for First Order Particle Transport Equations* (1 year)
- For the second summer as an intern and continued as part of PhD research three codes are written for the first order particle transport equation including a least squares finite element method, a discontinuous finite element method, and a mixed finite element method. These codes are written in one and two dimensions. Uncollided-flux preconditioners are used in conjunction with the linear solvers of each method. Iterative linear solvers are also written for these methods.
- Codes written in MATLAB
- *Preconditioning Mixed Finite Element Methods for Flow Equations in Porous Media* (1 year)
- Solutions of second order flow equations with diagonal permeability tensors are used as preconditioners for solutions of flow equations with full permeability tensors within a mixed finite element method. The method also incorporates the method of homogenization on the permeability tensors over the domain.
- Codes written in C++, MATLAB
- *GPU Finite Element Solvers for Transport Equations* (1 year)
- Several of the above transport codes will be written in OpenCL or CUDA to run on GPU's.
- Codes written in C++, OpenCL, CUDA

Previous Research

- *Matrix Preconditioning for Photon Transport Equations* (1 year)
- For the first summer as a technical intern at Sandia National Laboratories-Science of Extreme Environment Research Institute a multilevel preconditioning package, ML, developed by the Trilinos group, was used to speed up the linear solver within SCEPTRE, a radiation transport code also developed at Sandia. Specifically various

smoothers were tried within ML to speed up the conjugate gradient method within the SCEPTRE code. Chebyshev polynomials were found to work well on the given problem.

- Codes developed in C++
- *Finite Element Solvers for Computational Homogenization in Porous Media* (1 year)
- As part of a comprehensive examination, finite element solvers were written for second order fluid transport equations in one, two and three dimensions. The assumption of periodicity on the homogenized problem allows for a unique matrix storage structure that is easily applicable to iterative solvers.
- Codes written in MATLAB
- *Factor Analysis Approximations in Dimension Reduction for Face Recognition Software* (1 year)
- Centroid approximations were used as approximations to the singular value decomposition within face recognition software.
- Software was developed in MATLAB

- *Using Image Processing in Determining Wildlife Populations* (1 year)
- As part of a student team, an unsupervised object detection algorithm was developed to calculate wildlife populations from aerial images.
- Algorithm developed in MATLAB

Software Experience

- MATLAB (5 years)
- C++, OpenCL, CUDA (1 year)
- Java (1 year)

Publications

- M. Rigley, *Intermediate Complexity Biological Modeling Framework for Mountain Lakes Based on Physical Structure*, Masters Thesis, Utah State University, Fall 2009.
- M. Rigley, *Matrix Preconditioning for Photon Transport Equations*, technical report, SAND 2011-6529 P, Summer 2011.
- M. Rigley, *Matrix Preconditioning for Neutron Transport Equations*, technical report, SAND 2012-7676 P, Summer 2012.
- D. Sunderland, M. Garlick, M. Rigley, M. Scott, and K. Keepers, *Efficient Assay Algorithm for PCR Primers*, technical report, USU, April 2008.
- M. Rigley, C. Drumm, J. Koebbe, *Uncollided-Flux Preconditioning for the First Order Transport Equation*, Mathematics & Computation May 2013, Sun Valley Idaho.
- *Preconditioners and Mixed Finite Element Methods for Fluid Flow in Porous Media*, dissertation topic, to be completed Spring 2013.

Presentations

- Matrix Preconditioning for Photon Transport Equations (Poster), Sandia SIP Poster Event, August 2011, SAND number unknown.
- Transport Equations: Preconditioning Discontinuous FEM's, SAND 2012-6203P, SEERI (Science of Extreme Environments Research Institute) End of Summer Presentations, August 2012.
- Numerical Techniques in Modeling Fluid Flow Through Porous Media, Intermountain Graduate Research Symposium, April 2012.
- Intermediate Complexity Biological Modeling Framework for Mountain Lakes Bases on Physical Structure, Intermountain Graduate Research Symposium, March 2010.

- Show Your True Eigenface: A Workshop on Image Processing, Sponsored by MSPDAWG – USU’s Mathematics and Statistics Professional Development and Working Group, with funding from the Park City Mathematics Institute and Utah State University, May 2011.

Awards/Appointments

- Research Fellow, National Physical Science Consortium (2011-2013)
- Image Processing Summer School, Park City Math Institute (2009)
- Teaching Above and Beyond the Call of Duty, Math Department Award (2009)